6.3-4-5

AFCRL-63-185

RADIO CORPORATION OF AMERICA
RCA LABORATORIES

THEORY OF ADJUSTABLE SWITCHING NETWORKS

BY

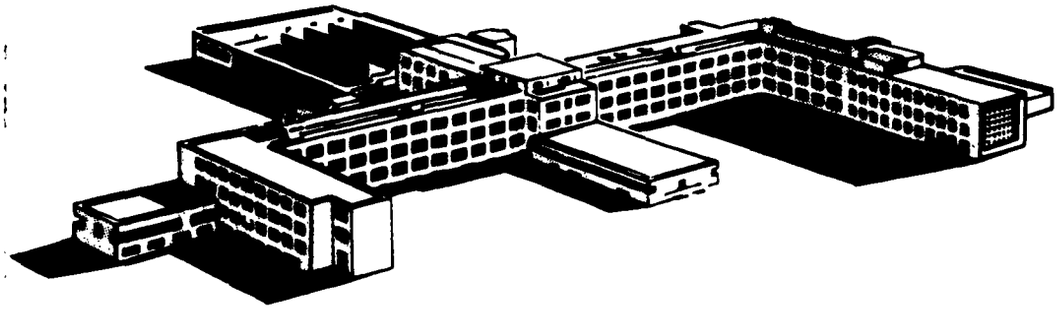
S. AMAREL
S. Y. LEVY
C. V. SRINIVASAN
R. O. WINDER

CONTRACT NO. AF19(604)-8423
PROJECT NO. 4641, TASK NO. 464101

SPECIAL SCIENTIFIC REPORT NO. 2

APRIL 30, 1963

PREPARED FOR
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS



DAVID SARNOFF RESEARCH CENTER
PRINCETON, NEW JERSEY

AFCRL-63-185

THEORY OF ADJUSTABLE SWITCHING NETWORKS

by

S. Amarel
S. Y. Levy
C. V. Srinivasan
R. O. Winder

RADIO CORPORATION OF AMERICA
RCA LABORATORIES
PRINCETON, NEW JERSEY

SPECIAL SCIENTIFIC REPORT NO. 2

APRIL 30, 1963

CONTRACT NO. AF19(604)-8423
PROJECT NO. 4641, TASK NO. 464101

PREPARED FOR
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS

ABSTRACT

This report discusses continuations of the work reported in Special Scientific Report No. 1 on this same contract. Part 1 (Threshold Logic) consists of a comprehensive survey of threshold logic, a geometric result relevant to estimating the number of threshold functions, generalizations and strengthening of known bounds on the logical capabilities of threshold gate networks, computer-aided work on the realization of arbitrary functions by networks of three-input majority gates (including a list of realizations for every type of four-argument switching function), and a comparison of two methods for synthesis of very large threshold gates: the well-known Bayesian approach and a geometric alternative. The latter method is shown to be preferable.

Part 2 (Reliability of Switching Networks) presents a survey of several important schemes for introducing redundancy into a combinational network for the improvement of reliability — comparisons are made with the recursive triangle system (see Special Scientific Report No. 1); some extensions of the previous analyses of recursive triangles (specifically, more general results on the "and" function, study of a nonsymmetric function, ABvCD, and generalization to the "nor" function); and initial results on the incorporation of memory and feedback to allow the use of fewer basic gates in a time-shared fashion.

PREFACE

The two major parts of this report cover the two branches of our investigation of the theory of adjustable switching networks; Threshold Logic and Reliability of Switching Networks. Each part thoroughly summarizes and analyzes the research performed during the last year. Research performed in the first year of the contract has been reported in Special Scientific Report No. 1; the present report should be considered a continuation of the latter, where definitions, motivations, historical background, etc., can be found. Work on the theory of adjustable switching networks is continuing.

A cumulative list of the Scientific Reports issued on this contract follows:

Scientific Report No. 1. Winder, R. O., "More About Threshold Logic", AFCRL 702, July 14, 1961.

Scientific Report No. 2. Brzozowski, J. A., "Reliability of Triangular Switching Networks with Intermittent Failures", AFCRL 785, August 14, 1961.

Scientific Report No. 3. Levy, S. Y., "Triangular Rectifier Networks", AFCRL 786, August 23, 1961.

Scientific Report No. 4. Miiller, H. S., Winder, R. O., "Majority Logic by Geometric Methods", AFCRL 792, July 13, 1961.

Scientific Report No. 5. Amarel, S., Cooke, G., Winder, R. O., "Majority Gate Networks", AFCRL 793, August 14, 1961.

Special Scientific Report No. 1. Amarel, S., Levy, S. Y., Winder, R. O., "Theory of Adjustable Switching Networks", AFCRL-62-318, April 30, 1962.

Scientific Report No. 6. Winder, R. O., "Threshold Logic in Artificial Intelligence", AFCRL-63-6, November 15, 1962.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

PART 1

THRESHOLD LOGIC

I. INTRODUCTION

Part 1 of Special Scientific Report No. 2 consists of several sections: Section II and Section VII (References, pg. 39) constitute a comprehensive survey of threshold logic. Sections III, IV, V, and VI describe the present state of several lines of investigation under this contract. (Whereas at the end of the first year of this contract several primary lines of investigation had terminated, so that a final-type report — Special Scientific Report No. 1, Part 1 — was written, this is not the case this year.) The Appendix of Part 1 (Section VIII) is a listing of networks which resulted from the investigations reported in Section V.

Threshold logic, especially as reported in Special Scientific Report No. 1, has been primarily motivated by an interest in computer applications — design with threshold gates. Recently, the interesting possibilities of a second major field of application have become apparent: To define complicated decision functions of many variables, students of artificial intelligence almost invariably have used the simple and natural idea of "linear separation" — their basic functions have turned out to be threshold functions. It seems that both biological and engineering reasons lie behind this: First, the psycho-physiological theories of learning and perception (e.g., Hebb) have employed theories of neural nets (McCulloch and Pitts); these neural models are, in fact, typical threshold gates. Second, investigators of self-adjusting switching devices have found neuron-like elements to be easily controllable.

Threshold logic has much to offer students of artificial intelligence. Furthermore, an increased interaction will surely produce interesting new developments in threshold logic. For example, in Section VI we consider an important common area between the fields: the synthesis of a single threshold gate for purposes of pattern recognition. The frequently used probabilistic approach to the problem is outlined, its main shortcomings discussed, and an alternative given, which arises from the switching theoretic point of view. It is shown that the probabilistic results are always worse than the crudest form of the switching theoretic results.

Section II provides a general survey of the relevant sections of threshold logic. A guide to the literature is provided, specifically slanted toward the student of artificial intelligence. This is an attempt to stimulate the interest of such people in threshold logic, to make the literature easily accessible, and to encourage their continued attention.

The problem of estimating the number of threshold functions, out of all switching functions of a given number of arguments, is an important one. It can be shown equivalent (see Special Scientific Report No. 1) to the geometric problem of partitioning Euclidean n-space by a certain collection of hyperplanes. In connection with this approach, Section III reports an important theoretical step toward a resolution of the problem. Work on application of the result continues.

In Section IV a generalization of earlier estimates (or more specifically, upper bounds) is made. This generalization gives upper bounds on the number of incompletely specified threshold functions: Suppose m of the possible 2^n input combinations to a gate have given specified outputs associated with them. Out of the entirety of 2^m such incompletely specified functions, averaging over the $\binom{2^n}{m}$ choices of the m points, less than m^n will be realizable with a single threshold gate. This, for example, puts severe restrictions on the logical capabilities of a network such as the Perceptron, where adaptation is possible on only one threshold gate. Section IV also strengthens an earlier result of Cameron: The random function of n arguments will require more than $\sqrt{2^{(n+1)}/n}$ threshold gates, interconnected in a network, for realization. A bound for the likely size of networks for realization of incompletely specified functions is also obtained.

Section V reports computer-aided investigations of the problem of realizing arbitrary functions with networks of simple three-input majority gates. Results are incomplete and inconclusive; much work remains to be done. An outcome of the experimentation, given in the Appendix (Section VIII), is a set of threshold gate network realizations for each type of four-argument switching function — the networks have been proved minimal in their number of stages.

The ideas of Section VI, mentioned above, have introduced two new lines of investigation, both of which will be studied in the third year of this contract. The first concerns basic limitations of the Bayesian approach to threshold gate design. The limitations stem from certain basic "independence assumptions", necessary for the Bayesian synthesis procedure. The first theorem of Section VI states that if a threshold function with equiprobable input combinations satisfies the independence assumptions, it must be a trivial function. A strengthening of this result is now being investigated: Any function, with any input distribution, that satisfies the independence assumptions, must be trivial. In other words, the Bayesian approach cannot be applied to nontrivial functions with any guarantee of success. Important consequences for artificial intelligence, and specifically for adaptive systems of threshold gates, are under study. The second new line of investigation concerns the switching theoretic countersuggestion to the Bayesian synthesis procedure, outlined in Section VI; it will also receive more attention. The development of heuristic programs for the synthesis of threshold gate networks is also contemplated.

II. SURVEY OF THRESHOLD LOGIC

by R. O. Winder

Threshold logic areas of interest to investigators of artificial intelligence are outlined, and some problems whose solutions would be significant in artificial intelligence are suggested in this section.

Most of the threshold logic literature deals with one or more of the following three areas: Conditions which functions must satisfy to be threshold functions, algorithms for determining the existence and nature of realizations, and methods (heuristic) for synthesizing networks built from threshold gates, often, in particular, from three-input majority gates (AB + AC + BC). In the following we discuss the large body of simple transformations and other properties known, then the necessary conditions, one-element test-synthesis, network synthesis, and finally the more important miscellaneous results, (see [Winder-3] for an earlier survey).

A. SIMPLE PROPERTIES

The following are typical simple facts about threshold functions:

1. Realizing weights and threshold are not unique.
2. All threshold functions can be realized with integral weights and threshold.
3. Any pair of (unequal) numbers (i.e., 0 and 1, -1 and +1, etc.) can be used as the numerical equivalents of the switching variables in order to define threshold functions. (The same weights can be used in different systems; the appropriate thresholds are easily calculated.)
4. Given a threshold function f, functions derived from f by permuting or complementing arguments, or by dualizing or complementing f, or any combination of these, are also threshold functions. (Again, appropriate transformation rules are easy to define.)

A general treatment of these, and most of the topics mentioned below, can be found in the papers [Elgot], [Gabelman-1], [Muroga-2, 5], and [Winder-1, 4]. These are relatively comprehensive papers, with much duplication of the basic material. [Coates-Lewis] also covers a wide range of material, but with a specific slant toward the test-synthesis problem.

B. NECESSARY CONDITIONS

The simplest condition which all threshold functions must meet is unateness* discussed originally in [Muroga-1], [McNaughton], and [Paull-McCluskey]. [Paull-McCluskey], [Winder-1], and [Muroga-2] generalize this condition, obtaining an infinite family of necessary conditions: 1-monotonicity (equivalent to unateness), 2-monotonicity, etc. A function satisfying all of these conditions is completely monotonic. [Winder-1, 4] give the most complete treatment of these ideas, including an important function due to E. F. Moore of Bell Laboratories -- a twelve-argument function which is completely monotonic, but is not a threshold function.

Because complete monotonicity fails to characterize threshold functions, this idea in turn is generalized in various ways in [Elgot] and [Gabelman-1]. [Winder-2, 4] discusses these generalizations further, settling some questions left unanswered in [Elgot] and [Elgot-Muroga], and concentrating on a second infinite family of necessary conditions: 2-assummability (equivalent to complete monotonicity), 3-assummability, etc. A function satisfying all of these conditions is proved in [Elgot] to be a threshold function. This necessary and sufficient condition was published about the same time in [Chow-1]. It is discussed in terms of convex sets in [Highleyman] and [Gabelman-2].

These various necessary conditions have obvious importance in artificial intelligence. Even for sparsely specified functions (where most inputs do not have outputs specified) of many arguments, functions can often be shown to be nonthreshold functions by simple observations. More work is needed, however, to adapt the ideas more specifically to the case of sparse specification, many inputs.

---

*A function is unate when it can be expressed by a Boolean expression in which each variable appears uniformly: everywhere with negation or everywhere without.

C. TEST-SYNTHESIS

The question: "Is a given function a threshold function,and if it is, what weight and theshold assignments realize it?" is easily seen to be equivalent to the consistency and solution of a certain system of linear inequalities I, on the weights and threshold as "unknowns" -- [McNaughton]. If the function has n arguments, I contains 2^n inequalities. The theory of k-monotonicities can be used to reduce this system to manageable proportions -- in various forms, most of the papers to be mentioned in this section make use of ideas equivalent to at least 1-monotonicity, and often 2-monotonicity. The problem of the equivalence of these reduced systems I' with the original is discussed in [Winder-1, 4]. Many different methods of solution for I' have been proposed. Algebraic solution, which involves successive elimination of unknowns and yields a specification of all solutions, is discussed in [Elgot] and [Winder-1, 4].

The important idea of solving I' by linear programming is discussed in [Minnick], [Stram-1], [Muroga-1], and [Einhorn]. Integers are usually obtained under a condition that their sum (absolute values) be minimum; by a technical device, integer programming has been avoidable except for some specially constructed examples of E. F. Moore and [Winder-2, 4] (thus settling the other open problem of [Elgot-Muroga]). A game theoretic approach to the solution of I' is discussed in [Akers-1]. Both the linear programming and game theoretic approaches produce just one realization, or else prove that there are none.

The procedure of [Coates-Lewis] and [Coates-Kirchner-Lewis] is a specialized algorithm which algebraically produces a single solution, or proves that there are none. A geometric, heuristic procedure is described in [Stram-1, 2]. In [Dadda-1, 2] methods suitable for small n are given. In [Varshavskii-1] a fallacious procedure is summarized.

The question of obtaining minimal integral realizations (the sum of absolute values of weights and threshold minimized) is very nicely solved in [Gabelman-1, 3 ; the procedure is refined and rigorously established in [Winder-4]. Synthesis when the function is incompletely specified is treated in [Winder-1, 4] (algebraically, with not too many inputs unspecified) and in [Singleton] (by matrices, with not too many inputs specified). The intermediate problem has obvious importance in artificial intelligence and in character recognition; the ideas described in [Winder-6] (and section VI of this report) may provide a start in this direction.

All methods discussed above (except those of [Winder-6]) are switching-theory oriented, in the sense that they work exactly, and are not feasible when the number of arguments becomes large (over 20, say). It is likely that artificial intelligence motivations will soon be producing some useful results for large n (and for largely unspecified functions).

D. NETWORKS

The question of synthesizing a given function in a network of threshold gates is a typically difficult switching theoretic problem. (A single threshold gate realizes a threshold function; a network of threshold gates corresponds to a composition of threshold functions.) It has been considered under various constraints: In computer design applications, tolerance considerations drastically limit the number of inputs that can be allowed. [Muroga-4] is a general discussion of this; [Muroga-5] goes into important details. There are several papers on synthesis by networks of three-input majority gates (AB + AC + BC), the simplest threshold functions beyond the conventional NOT, OR, AND, and their variants. The methods can be divided according to the means of function representation: Working algebraically, [Lindaman-1,2], [Cohn-Lindaman], and [Akers-2] discuss various transformations, analogous to the ordinary Boolean transformations, which may be helpful in synthesis. [Winder-4] gives an axiomatic treatment which may be helpful in synthesis. [Winder-4] gives an axiomatic treatment which may be more useful in this context than that of [Cohn-Lindaman]. A geometric approach is described in [Miller-Winder], and results on all four-argument functions are reported in [Winder-5] (and in Section VI of this report). [Akers-3] employs truth tables in interesting fashion.

The synthesis of a specific class of functions -- symmetric functions -- has been treated in [Muroga-2], [Minnick], and [Kautz].

A geometric approach to the unrestricted problem is described in [Winder-2, 4], where the question of determining whether a given function can be realized using just two threshold gates is answered. An algebraic method, where constraints on the magnitude of the threshold can be made conveniently, is given in [Lewis-Coates]. [Varshavskii-2] summarizes an interesting construction, and gives a startling (but false, see below) "theorem" that a function of n arguments can always be realized in a network of n + 1 threshold gates. Many of the one-element

test-synthesis procedures discussed above go on, when a function is shown to be nonthreshold, to produce a network in heuristic fashion. Typically, several fragmentary functions are OR'ed together. A heuristic modification of linear programming is used in [Minnick] to produce a table of network realizations for all (classes of) 4-argument functions.

The network synthesis is just beginning, and so far is restrained to very small n (less than 10). Perhaps the most promising approach for the (heuristic) design of large-n two-level networks (several "upper-level" gates feeding a single "output gate", as in the Perceptron, etc.) lies in the methods of [Winder-2, 4], where failures of the k-monotonicity tests are used to specify appropriate upper-level gates -- i.e., each upper gate can make the final decision as a threshold function. (Incidentally, the conjecture that the output gate can always be taken as a simple many-input majority gate, still yielding optimal networks, can be disproved by counterexample.)

E. OTHER MAJOR SUBJECTS

The parameters of [Chow-2] are used implicitly in Section VI. Besides their practical use, the paper cited opens up a theoretically very interesting possibility that threshold functions may be characterizable by their parameters [the $m(f_{x_i})$ and $m(f)$].

A large number of "functional properties" of threshold functions, dealing with various combinations of functions, chains of functions obtained by varying threshold, duality relations, classification by prime implicants, and similar ideas, are treated in [Muroga-2, 3, 6], [Elgot], and [Gabelman-1]. These ideas are rather of theoretical than practical interest, and at present do not appear to have much bearing on artificial intelligence.

R_n, the number of threshold functions of n arguments, has been shown to be bounded above ([Muroga-2] [Winder-1 4], [Cameron]) and below ([Goto-Takahasi], [Muroga-2, 6]) as follows:

$$2^{0.33n^2} < R_n < 2^{n^2} << 2^{2^n} \quad (n > 1),$$

where 2^{2^n} is the total number of switching functions of n arguments. Thus threshold functions become a vanishingly small proportion of all functions, as n increases. [This suggests that the often-used idea of randomly chosen upper-level threshold gates in a 2-level network is likely to require the output gate to realize a nonthreshold function. A rational procedure, such as that mentioned in the section on networks, is much more likely to work with fewer upper-level gates. In particular, tentative ideas indicate that the output gate, not the upper-level gates, should be partially specified at the outset (for example, the weights' signs, and possibly their relative absolute magnitudes, might be determined on a heuristic or random basis); then existing methods, as mentioned above, can be used to specify appropriate upper-level gates (and the output gate), even for very sparsely specified, many-input functions. (Modifications for iterative synthesis would be required.) As noted earlier, the best networks are obtained by allowing the output gate to be as complicated as is needed.] In [Cameron] it is shown, using the upper bound, that networks to realize given n-argument functions will for most functions require exponentially many threshold gates -- something like $2^{n/3}$. Bounds on the size of integral weights that will be required for some functions are found in [Myhill-Kautz] and [Muroga-5].

In Section IV of this report, generalizations of these results are made: Bounds on the number of imcompletely specified functions realizable by a single threshold gate are obtained, and are applied to indicate fundamental limitations of the Perceptron-Pappa-Pandemonium type networks of threshold gates. Cameron's bound, $2^{n/3}$, is improved to $\frac{1}{n} 2^{\frac{n+1}{2}}$. In Section III a theoretical geometric result is obtained which may lead to further improvement of the various bounds.

The classification and enumeration of classes, for threshold functions, have been studied in [Winder-1,4] and [Muroga-1]. Tables listing representatives of all six-argument threshold functions, with minimum integral realizations, are to be found in [Winder-4] and [Muroga-6]. A more efficient classification scheme has recently been described in [Goto-Takahasi]: relevant results are in [Muroga-5].

Two variants of threshold logic are discussed in [Ercoli-Mercurio] and [Hotz].

III. PARTITIONS OF N-SPACE BY HYPERPLANES

by R. O. Winder

The following problem has arisen in the field of switching theory (i.e., computer-motivated mathematical logic): Given a well-defined set of m (n-1)-dimensional linear subspaces in n-dimensional Euclidean space, all passing through some given point, to find the number of regions into which the n-space is divided by the m "hyperplanes". We will develop below a formula for this quantity which seems to be of general mathematical interest.

The best result to date has been a formula for the special case of the hyperplanes lying in "general position" (in a sense which will be made explicit later). The hyperplanes of the original problem failed to satisfy this condition -- the formula provided, then, an upper bound on the desired quantity. The formula is ("B" for "bound")

$$B_n^m = 2 \sum_{i=0}^{n-1} \binom{m-1}{i}. \qquad (1)$$

See* [Winder-1], [Cameron].

> Definition: A set of k (n-1)-dimensional "hyperplanes" in n-dimensional Euclidean space, all planes passing through some given point, is nondegenerate if the intersection (a linear subspace of generally lower dimension) has dimension n-k. If the dimension is of different parity than n-k, the set is odd-degenerate; if the dimension has the same parity as n-k, even-degenerate.

The term "general position", used above, is defined by the property that every set of n or fewer of the given m planes should be nondegenerate.

> Theorem: The number of regions into which m hyperplanes, all passing through some common point, divide n-space is equal to the number of distinct even-degenerate subsets of the given m planes minus the number of distinct odd-degenerate subsets. (The empty subset is included and is counted as even-degenerate.)

Example: Suppose that the set of m planes are in general position. Then the number of regions is

---

*D. T. Perkins, D. G. Willis, and E. A. Whitmore, of Lockheed (Missiles and Space Division), have not published their early results on this problem.

$$N_n^m = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{n} - \binom{m}{n+1} + \binom{m}{n+2} - \binom{m}{n+3} + \cdots \pm \binom{m}{m}, \quad (2)$$

because each subset of n or fewer hyperplanes, counted in the first (n+1) terms, are nondegenerate, and so even-degenerate, and thus are counted positively in the sum. All subsets of more than n hyperplanes have just a point in common, i.e., their intersection has dimension 0. Thus we have the alternating signs for the remainder of the expression. But

$$\sum_{i=0}^{m} \binom{m}{i} (-1)^{n+i+1} = (1-1)^m = 0. \quad (3)$$

Adding corresponding terms in (2) and (3), we have

$$N_n^m = 2\left[\binom{m}{n-1} + \binom{m}{n-3} + \binom{m}{n-5} + \cdots\right] \quad (4)$$

where the last term is $\binom{m}{1}$ or $\binom{m}{0}$. But

$$\binom{m}{i} = \binom{m-1}{i} + \binom{m-1}{i-1} \; ; \quad (5)$$

so substituting in (4) (except for $\binom{m}{0}$ if it's present), we obtain B_n^m as given in (1).

Proof of theorem: We proceed by induction on m. For m=1, there are two subsets -- the empty set and the given hyperplane, each of which are trivially nondegenerate, and so even-degenerate. Thus the theorem would predict two regions, which is, of course, correct.

Supposing the theorem verified for sets of m or fewer hyperplanes, we consider a set of (m+1) distinct hyperplanes ($m \geq 1$), $H_0, H_1, \ldots, H_m$, all passing through some common point. Suppose we denote the number of regions that they divide n-space into by

$$\#\left\{ H_0, H_1, \ldots, H_m \right\}^n .$$

We now select one of the planes, say H_0, and will prove the following:

$$\#\left\{ H_0, H_1, \ldots, H_m \right\}^n = \#\left\{ H_1, H_2, \ldots, H_m \right\}^n + \#\left\{ H_0H_1, H_0H_2, \ldots, H_0H_m \right\}^{n-1} \qquad (6)$$

where H_0H_i represents the intersection of the two planes, and the right-hand expression of (6) represents a subdivision of the hyperplane H_0 (an (n-1)-dimensional Euclidean space in its own right) by its intersections with the other planes (since the hyperplanes are distinct, and since no pair of hyperplanes is parallel, the intersections H_0H_i are, indeed, (n-2)-dimensional "hyperplanes" in H_0). Although the hyperplanes $H_0, H_1, \ldots, H_m$ are distinct, the entities $H_0H_1, H_0H_2, \ldots, H_0H_m$ may not be; when we enclose them by brackets we are thinking of the set, which may then have less than m members.

Justification of (6) will not be completely rigorous; we rely on the reader's intuition: Consider n-space as partitioned by the hyperplanes $H_1, H_2, \ldots, H_m$. When a new hyperplane H_0 is introduced, how many regions are affected? Clearly, only those into which H_0 enters, and enters to the extent that it has an (n-1)-dimensional intersection with them. Each such region is clearly cut into two pieces, since the regions are simply connected (they are bounded by hyperplanes) and the separation is by a hyperplane. Thus the new number of regions is just the old, plus the number of (n-1)-dimensional regions into which H_0 is cut by intersections with the original hyperplanes. (To say it slightly differently: Each piece into which H_0 is cut acts as a divider in some one of the original regions, thus adding one region to the previous total.) This establishes (6).

Now for the induction step: By hypothesis, the middle term of (6) is equal to the number of even-degenerate subsets of $\left\{ H_1, \ldots, H_m \right\}$ minus the number of odd-degenerate subsets. This accounts for all subsets of $\left\{H_0, H_1, \ldots, H_m\right\}$ that don't involve H_0; the induction step will be complete if we can show that the number of even-degenerate subsets containing H_0, minus the number of odd-degenerate subsets containing H_0, equals the final term in (6). But by hypothesis, the final term is equal to the number of even-degenerate subsets of

$\{ H_0H_1, \ldots, H_0H_m \}$, in H_0, minus the number of odd-degenerate subsets. Suppose we number the H_i so that H_0H_1, H_0H_2, ..., H_0H_p are distinct, while each of H_0H_{p+1}, ..., H_0H_m is a duplicate of one of the first p. Now consider any subset $\{ H_0H_{i_1}, H_0H_{i_2}, \ldots, H_0H_{i_k} \}$ where each $i_j \leq p$. The intersection

$$(H_0H_{i_1})(H_0H_{i_2}) \cdots (H_0H_{i_k}) = H_0H_{i_1}H_{i_2} \cdots H_{i_k}. \tag{7}$$

Note that the degeneracy (odd or even) of the subset $\{ H_0H_{i_1}, \ldots, H_0H_{i_k} \}$ in (n-1)-space is the same as the degeneracy of the subset $\{H_0, H_{i_1}, H_{i_2}, \ldots, H_{i_k}\}$ in the original n-space (because in the latter case the spatial dimension is one greater, and the number of planes is one greater, while the intersection -- and its dimension -- remains unchanged, by (7)). We have proved this equivalence for all subsets of $\{ H_0H_1, \ldots, H_0H_p \}$ since here there are no duplicates. Thus, the right-hand side of (6) equals the number of even-degenerate subsets of $\{ H_0, H_1, \ldots, H_m \}$ <u>except</u> those containing H_0 and at least one of $H_{p+1}, \ldots, H_m$, minus the number of odd-degenerate subsets with the same exception. If we can show that the sum corresponding to this exception is zero, the proof will be complete.

So consider H_{p+1}. By assumption, for some $r \leq p$

$$H_0H_{p+1} = H_0H_r . \tag{8}$$

Consider all subsets of $\{ H_0, H_1, \ldots, H_m \}$ which contain H_0 and H_{p+1} but none of $H_{p+2}, \ldots, H_m$. They come in pairs; for each such subset containing H_r there is one identical except lacking H_r. Consider the intersections

corresponding to such a pair -- call them $H_0 \sqcap H_{p+1}$ and $H_0 \sqcap H_r\ H_{p+1}$. ("$\sqcap$" for some "product" of hyperplanes). Multiplying (i.e., intersecting) each side of (8) by $\sqcap H_{p+1}$ we see that the intersections are equal:

$$H_0 \sqcap H_{p+1} = H_0 \sqcap H_r\ H_{p+1}. \tag{9}$$

Thus the dimensions are equal. And since the number of hyperplanes in the corresponding subsets differ by one (H_r), one of the subsets must be even-degenerate, and the other odd-degenerate. Because of this cancellation, we can strengthen the next-to-last sentence of the previous paragraph: The right side of (6) equals the number of even-degenerate subsets of $\{ H_0, H_1, \ldots, H_m \}$ except those containing H_0 and at least one of $H_{p+2}, \ldots, H_m$, minus the number of odd-degenerate subsets with the same exception.

Now we repeat the argument with H_{p+2}, then again with H_{p+3}, and so on, until the exceptions are removed: The left side of (6) equals the right side of (6), which equals the number of even-degenerate subsets of $\{ H_0, H_1, \ldots, H_m \}$ minus the number of odd-degenerate subsets, QED.

IV. BOUNDS ON THRESHOLD GATE REALIZABILITY

by R. O. Winder

It is well known* [Winder-1], [Cameron] that the number of threshold functions of n arguments is less than

$$B_n \equiv 2\sum_{i=0}^{n} \binom{2^n-1}{i} .$$

Using this bound, Cameron [op cit] shows that the number of n-argument functions realizable by a network of at most k threshold gates is (asymptotically) less than 2^{k^3} . As Cameron pointed out, this implies that at least one switching function of n arguments (and probably most of them) requires more than $2^{n/3}$ threshold gates for realization. The purpose of this note is to generalize these results to functions incompletely specified; an improvement in the asymptotic bound $2^{n/3}$ will also be obtained. Applications in character recognition and self-organizing systems are discussed.

The basic bound B_n is derived from the following basic lemma (see [Cameron] for a good discussion of its proof — the proof in [Winder-4] is virtually identical, but less well explained):

> Lemma: If m hyperplanes are passed through the origin of an (n+1)-dimensional Euclidean space, the space is divided into a number of regions at most
>
> $$B_n^m \equiv 2\sum_{i=0}^{n} \binom{m-1}{i}.$$

The bound B_n is then obtained by considering an (n+1)-dimensional "realization space" — the space consisting of points $\vec{a} = (a_0, a_1, \ldots, a_n)$, each of which represents the realization of some threshold function (a bias and n weights). (We assume a ± 1 logic.) By taking all possible choices of sign, we consider 2^n hyperplanes

$$a_0 \pm a_1 \pm a_2 \pm \ldots \pm a_n = 0.$$

Two points in the realization space represent the same function if and only

*D. T. Perkins, D. G. Willis, and E. A. Whitmore, of Lockheed (Missiles and Space Division), have not published their early results on this problem.

if they are not separated by any of these hyperplanes. Thus the regions, with boundaries defined by these hyperplanes, correspond one-to-one with threshold functions. Thus setting $m = 2^n$ in the lemma gives B_n.

Now: Suppose we select exactly m out of the 2^n possible input combinations. How many switching functions, no two of which agree in value on all m of these points, can be realized by a single threshold gate? Clearly, by the same argument, there are at most B_n^m. (Because the m points correspond to m of the hyperplanes, and again, we are asking how many regions the realization space is divided into, here by m hyperplanes.)

Bounds on B_n^m are easily obtained:

$$2\sum_{i}^{n}\binom{m-1}{i} = 2\binom{m-1}{n}\left[1 + \frac{n}{m-n} + \frac{n(n-1)}{(m-n)(m-n+1)} + \cdots + \frac{n!}{(m-n)\cdots(m-1)}\right]$$

$$< 2\binom{m-1}{n}\left[1 + \frac{n}{m-n} + \frac{n^2}{(m-n)2} + \frac{n^3}{(m-n)3} + \cdots\right]$$

$$= 2\binom{m-1}{n}\frac{1}{1-\frac{n}{m-n}} = 2\binom{m-1}{n}\frac{m-n}{m-2n}$$

(providing $n < m-n$, i.e., $m > 2n$, which we henceforth assume). So

$$B_n^m < 2\,\frac{(m-1)(m-2)\cdots(m-(n-1))(m-n)}{n!}\cdot\frac{m-n}{m-2n}\,.$$

Assuming $m \geq 3n + 2$ and $n \geq 2$, which we do henceforth, we take four of the right-most factors above:

$$\frac{(m-(n-1))(m-n)(m-n)}{m-2n} = (m-(n-1))\left(m + \frac{n^2}{m-2n}\right)$$

$$\leq (m-(n-1))\left(m + \frac{n^2}{n+2}\right)$$

$$= m^2 - \frac{n^2+n-2-n^2}{n+2}\,m - \frac{n^2(n-1)}{n+2}$$

$$< m^2.$$

So

$$B_n^m < \frac{2m^n}{n!} < m^n$$

(we use the relatively sloppy bound m^n because of its convenience; sharper results below can be obtained using $2m^n/n!$). Summarizing:

Theorem: When exactly m points only are considered, at most

$$B_n^m < \frac{2m^n}{n!} < m^n$$

essentially different functions can be realized with a single threshold gate.

Rule-of-thumb: When $m > n \log_2 m$, single-gate realizability is improbable.

Justification: The total number of switching functions specified on exactly m points is 2^m. When $2^m > m^n$, we have that all such functions cannot be single-gate realized. Because of the rather generous approximations made in deriving the bounds, it seems reasonable to say that by this point, most functions cannot, in fact, be single-gate realized; hence, the rule-of-thumb. (Strictly speaking, we can only say that when $2^m/_m n > \alpha$, i.e., $m > n \log_2 m + \log_2 \alpha$, we have at best a $1/\alpha$ chance of realizing a random function, specified on m points. Taking $\alpha = 1000$, for instance, however, affects the application very little.)

An example: Suppose we have a Perceptron-Pappa-Pandemonium type network of threshold gates, i.e., a large number of randomly chosen threshold gates (A units) accepting inputs from some "retinal" field, and supplying, say, n = 100 inputs to an output threshold gate (the R-unit). Although the first level of gates is fixed, we are free to choose the weights and threshold of the final output gate as we please. What chance, then, have we of distinguishing between two given different types of pattern on the retinal field? Applying the rule of thumb, we see the following: If there are m = 1000 or more (out of the total of $2^{100} \sim 10^{30}$) input configurations in the two patterns, upon which we want the network to discriminate reliably, we probably will not be able to find any choice of weights and threshold that will work.

A valid objection to this type of argument is the following: The actual patterns that we want, in fact, to discriminate between, are not random — they are likely to have natural regularities and redundancies, which may make them more easily realizable than these enumerational statistics would indicate. However, in the case of Perceptrons, etc., note that the first level of randomly connected gates has served to eliminate such regularities! The Perceptrons, etc., therefore are at the mercy of these statistics; tne possibility of true generalization, where surely $m \gg 1000$ is required, is remote indeed.

Up to this point we have been considering single-gate synthesis. Suppose we now investigate the capabilities of networks of threshold gates. Let k be the number of gates in a given network, n and m defined as before, relative to the overall function produced by the network. Generalizing and strengthening Cameron's [op cit] arguments, we can prove

> Theorem: When exactly m points only are considered, a network receiving n external inputs, and consisting of k threshold gates, can realize at most
>
> $$m^{[nk + k(k-1)/2]}$$
>
> essentially different functions.

Proof: We assume that there are no feedbacks in the network, so that there is some order of gates: 1^{st}, 2^{nd}, ..., k^{th}. The 1^{st} receives just the externally provided n inputs, the 2^{nd} receives these plus the output of the 1^{st}, etc., and the k^{th} receives the basic n plus the output of the k-1 earlier gates, and supplies the network output. On the m specified input configurations, the 1^{st} gate realizes at most m^n threshold functions. For a given choice of function, the 2^{nd} gate is also presented with at most m specified input configurations, and so can realize at most $m^{(n+1)}$ functions. And so on: The k^{th} gate can realize at most $m^{(n+k-1)}$ functions. The total number of functions calculable by the network (with regard to the originally chosen m points) is, then, at most

$$m^n \cdot m^{n+1} \cdot \ldots \cdot m^{n+k-1} = m^{[nk + k(k-1)/2]}.$$

END OF PROOF.

Rule-of-Thumb: When

$$m \leq [nk + k(k-1)/2] \log_2 m,$$

a network of k gates will not suffice for most functions.

Proof is as for the previous rule-of-thumb.

An example: Consider a character-recognizing system with n = 100, and suppose m = 8000, i.e., 8000 variations on a given character are to be recognized. By the above, then, one would expect to require k = 6 threshold gates for realization, at least. Of course, here the well-behavedness of naturally arising functions can be expected to help; the bound is only a rough approximation of what might be needed.

Corollary: For completely specified functions ($m = 2^n$), if

$$k < -\frac{1}{2} - n + \sqrt{\left(\frac{1}{2} - n\right)^2 + \frac{2^{n+1}}{n}} \quad ,$$

then realizability in a network of k gates is improbable.

Proof: Substitute $m = 2^n$ in the above rule-of-thumb and solve for k. The bounds hold from n = 4:

n	k	n	k	n	k
4	1	10	8	16	77
5	2	11	12	17	109
6	2	12	18	18	154
7	3	13	26	19	218
8	4	14	37	20	305
9	5	15	54		

and above n = 20, the bound is very close to the dominant term: $\frac{1}{\sqrt{n}} 2^{\frac{n+1}{2}}$.
(This is a substantial improvement over Cameron's asymptotic bound of $2^{n/3}$.)

V. NETWORKS OF 3-INPUT MAJORITY GATES

We report here on an introductory exploration of synthesis methods for networks of 3-input majority gates. The results obtained were largely negative, so that the discussion will be brief.

A 3-input majority gate (called henceforth a 3-gate) realizes the switching function

$$\text{maj}\ (x,\ y,\ z) = xy + xz + yz = (xyz)$$

denoted, in this report, by the parenthetical closure of three arguments (as on the right). The synthesis problem is: Given an arbitrary switching function, find a network of 3-gates which realizes it. Minimality criteria may include the number of gates and the depth of the network (i.e., the maximal stage delay).

It was decided to study first functions of only four arguments. Computer programs (for the RCA 501) were written, debugged, and run to experiment with various trial synthesis procedures. The first of four programs written converted the Harvard Computation Laboratories list of 238 four-argument PN symmetry types into a list of easily manipulatable truth table representations in the RCA 501. Subsequent programs used this data, so that a trial procedure could be tried out (in effect) on every one of the 65,536 four-argument switching functions.

Program number two was written to try out a simple "synthesis-by-expansion" idea, just to see how it works. The expansion theorem:

$$P_x = (yx\ P_{\bar{y}})\ (y\bar{x}\ P_y)\ \bar{y}$$

was taken from [Cohn-Lindaman]. Its effect is to replace the problem of synthesizing P_x by the problems of synthesizing $P_{\bar{y}}$ and P_y separately, where y can be chosen so as to make these problems simpler, — all at the expense of three 3-gates. Program number two made the following ten substitutions for x and y:

$$(x,y) = (w,0),\ (w,x),\ (w,y),\ (w,z),$$
$$(x,0),\ (x,y),\ (x,z),\ (y,0),\ (y,z),\ (z,0).$$

Thus in each case P_y and $P_{\bar{y}}$ had only three arguments, with known optimal realizations. The number of gates required for each substitution for each function type was printed out. Choosing the best of the ten for each type, an average of 6.2 3-gates was required; this is surprisingly good, although far from the optimal networks obtainable. At most four stages are required by this expansion — all four were usually needed. The experiment showed that all ten of the basic expansions should be considered — the range from best to worse over this choice was large (e.g., from 6 to 11 gates were required, depending on the choice of expansion variable).

Program number three was written and run to find all two-stage 3-gate realizations that exist for 4-argument symmetry types. A modified form of the method of [Miiller-Winder] was used. Of the 238 symmetry types 52 were found so realizable. These results gave known optimal networks, then, for a large number of the functions we were studying.

Program number four was our first serious try at a general synthesis method; it was intended to apply to functions of any number of arguments. The algorithm is as follows:

(i) Initialize by listing all functions of the given input variables that can be realized by a single 3-gate. Then go through the following steps repetitively until a realization is found, or patience is exhausted.

(ii) Choose the k functions in the list which are most similar to the given function to be realized (i.e., which agree for the most input specifications). [If any agree exactly, exit.]

(iii) Take all possible triples from this set of k, forming the majority of each triple, and listing the resulting realized functions to replace the previous list of candidates. Return to step (ii).

The number of gates in the final network (when one is obtained) is $(3^{m+1}-1)/2$, where m recursions were required. The results of the application of this algorithm to the 238 4-argument symmetry types were disappointing. Unfortunately, the experimentation ended before the reason for this was satisfactorily established (bugs in the program? faulty intuition? too small a k used? — we used k = 10. etc.).

To complement the positive results of program number three (the 2-stage realizations) and to round out the study of four-argument functions, a long and tedious hand application of modifications of the [Miiller-Winder] method was made. Instead of treating each of the 238 Harvard PN symmetry types, we used a new and better classification method of [Goto-Takahasi]. By this method, only 83 types, called SD (for self dual), are needed. The hand calculations involved first a derivation of a set of 83 SD representatives (which are listed in the appendix) and then a realization of each, in an attempt to obtain 3-stage realizations for all 4-argument functions. (The average number of gates needed was 6.9, 2.8 stages average.) All but two SD types were 3-stage realizable (the realizations are listed in the appendix.) The two exceptions were the parity function and the "almost-parity" function (differing from the parity function at exactly one point).

It was found possible to prove that neither of these functions are 3-stage realizable, by the following argument: If a function f is 3-stage realizable, then f = maj(g,h,i), where each of g, h, and i are 2-stage realizable. The results of program number three identified the 19 SD types which are 2-stage realizable; these, then were the only candidates for g, h, and i. The next step in the argument is to note that since f agrees with the majority of g, h, and i at each of the 2^n input points (there are four arguments, and the representative SD types have five arguments, so n = 5), the sum of the number of agreeing points between f and g, f and h, and f and i, respectively, must be at least 2×2^n. By considering the 17 cases, however, it was found that none agreed with the parity function in more than 20 positions, so the sum could not exceed $3 \times 20 < 2 \times 2^5$. Thus the parity function is not 3-stage realizable. Similarly, only two 2-stage realizable types were found which agreed with the almost-parity function in more than 20 places; and they agreed in only 22 positions, so both would be needed to realize it. But by [Miiller-Winder] methods they did not suffice. Thus neither the parity function nor the almost-parity function can be 3-stage realized by 3-gates. Since we know all functions 2-stage realizable (by program number three), we have proved that the realizations listed in the appendix are minimal in number of stages (if not number of gates). Summarizing (these data were reported in [Winder-5]):

Minimal Number of Stages	Number of SD types	Number of Functions
0	1	10
1	1	80
2	17	10,260
3	62	55,152
4	2	34
	83	65,536

General conclusions: The study has only just begun. Expansion methods deserve more attention. Work on more ambitious general programs should commence with an analysis of program number four, and further experimentation with it. The realizations obtained by hand should be analyzed. Comparison with the methods of [Akers-3] should be made. (Akers has run ten of the 83 types with his program, obtaining two realizations definitely better than our hand calculations, two realizations definitely worse, two identical, and four that use fewer gates but more stages. Over these ten, he averages 4.7 gates, 3.4 stages, as compared with 5.9 gates by the simple expansion program (number two) and somewhat less than four stages, and compared with 5.5 gates by hand, and 2.7 stages.)

VI. TWO METHODS OF THRESHOLD GATE SYNTHESIS

by R. O. Winder

The optical recognition of characters is a commercially important problem; much of what is said below is significant in this area. Pattern recognition, a generalization of character recognition, plays many roles in artificial intelligence. As Minsky* sees it, for instance, one wants a computer to categorize the various problems presented to it into patterns, in order to choose suitable methods of solution. To do this, the computer determines, for each incoming problem, whether or not it satisfies each of a list of properties. (The analogous procedure is usually followed in optical character recognition.) If we call this list of properties an input vector, then the final step in the process is: Given an input vector, to determine into which category the given problem is to be placed; each category has a set of associated methods to be used in solving the problem. Without loss of generality, we will restrict ourselves to the question of determining whether or not the given input vector belongs to a single given pattern.

This reduces the problem to switching theory: To accomplish the last step in pattern recognition, a certain switching function of the input vector must be computed. (If the input vector belongs to the pattern, the value of the function for that vector is 1, otherwise 0.) Theoretically, switching theory could be used to realize such a function. However, because the number of inputs is often far too large to be handled by conventional switching theory, and because the function cannot be completely specified in practice, the following procedure is often followed to design a decision network. (We follow Minsky's excellent discussion of the matter.*)

A. BAYES NETS

We make several assumptions. First, we "assume that the situation is basically probabilistic" (Minsky). Next, we assume that we know, or can estimate, certain conditional probabilities: the probability of the i^{th} property holding, given that the input vector belongs to the pattern. Finally, we assume that these probabilities are, in a sense described below, independent. Now we can design a system which is able to compute the probability that a given input vector should be assigned functional value 1 or 0 (in or out of the pattern). When several possible patterns are being considered, the system might classify by choosing the pattern with the highest probability. More generally, we specify levels of confidence, so that the machine may fail to classify certain unusual problems; in our case of a single pattern, a confidence level $\Lambda > 1/2$ would be used.

---

* M. Minsky, "Steps Towards Artificial Intelligence", Proc. Inst. Radio Engrs. Vol. 49, pp. 8-30, January 1961.

More explicitly, suppose the inputs are associated with variables

$$\vec{x} = (x_1, x_2, \ldots\ldots, x_n)$$

and a particular state of the inputs is represented by

$$\vec{\xi} = (\xi_1, \xi_2, \ldots\ldots, \xi_n).$$

We are assuming that we know the probabilities

$$p_i = \Pr(x_i = 1 \mid f(\vec{x}) = 1)$$

$$r_i = \Pr(x_i = 1 \mid f(\vec{x}) = 0)$$

and

$$p_0 = \Pr(f(\vec{x}) = 1).$$

From these we obtain the related probabilities

$$1 - p_i = \Pr(x_i = 0 \mid f(\vec{x}) = 1)$$

$$1 - r_i = \Pr(x_i = 0 \mid f(\vec{x}) = 0).$$

The independence assumption is that

$$\Pr(\vec{x} = \vec{\xi} \mid f(\vec{x}) = 1) = \prod_{i=1}^{n} \Pr(x_i = \xi_i \mid f(\vec{x}) = 1)$$

and similarly with $f(\vec{x}) = 0$.

Now we use the Bayes law, writing

$$\Pr(f(\vec{x}) = 1 \text{ and } \vec{x} = \vec{\xi}) = \Pr(f(\vec{x}) = 1 \mid \vec{x} = \vec{\xi}) \cdot \Pr(\vec{x} = \vec{\xi})$$

$$= \Pr(\vec{x} = \vec{\xi} \mid f(\vec{x}) = 1) \cdot \Pr(f(\vec{x}) = 1).$$

Thus

$$\Pr(f(\vec{\xi}) = 1) = \Pr(f(\vec{x}) = 1 \mid \vec{x} = \vec{\xi})$$

$$= \Pr(\vec{x} = \vec{\xi} \mid f(\vec{x}) = 1) \cdot \frac{\Pr(f(\vec{x}) = 1)}{\Pr(\vec{x} = \vec{\xi})}.$$

Three alternatives are available: 1) We can write $Pr(\vec{x} = \vec{\xi})$ in terms of the probabilities given us*, yielding somewhat more complicated rules than we obtain below, 2) We can follow Minsky's discussion, taking at this point a maximum over the quantities

$$Pr(\vec{x} = \vec{\xi} \mid f(\vec{x}) = 1) = \prod_i Pr(x_i = \xi_i \mid f(\vec{x}) = 1)$$

for the various patterns (f's), or 3) We can assume at this point that the $\vec{\xi}$ are equiprobable, so that $Pr(\vec{x} = \vec{\xi})$ is independent of ξ. To simplify the illustration and future computations we chose the last alternative.

If $Pr(f(\vec{\xi}) = 1)$ is calculated to be greater than some confidence level, then, we will ask the machine to guess that $f(\vec{\xi}) = 1$ (i.e., to classify an incoming problem into the corresponding category). Otherwise, $f(\vec{\xi}) = 0$. We convert this calculation into the characteristic threshold function form as follows:

$$Pr(f(\vec{\xi}) = 1) \geq \Lambda \qquad \text{if and only if}$$

$$Pr(\vec{x} = \vec{\xi} \mid f(\vec{x}) = 1) \geq \Lambda \cdot \frac{Pr(\vec{x} = \vec{\xi})}{Pr(f(\vec{x}) = 1)} \qquad \text{if and only if}$$

$$\prod_i Pr(x_i = \xi_i \mid f(\vec{x}) = 1) \geq \Lambda \cdot \frac{Pr(\vec{x} = \vec{\xi})}{Pr(f(\vec{x}) = 1)} \qquad \text{if and only if}$$

$$\prod_i \frac{Pr(x_i = \xi_i \mid f(\vec{x}) = 1)}{Pr(x_i = 0 \mid f(\vec{x}) = 1)} \geq \theta \qquad \text{(for appropriate } \theta\text{) if and only if}$$

$$\sum_i \log \frac{Pr(x_i = \xi_i \mid f(\vec{x}) = 1)}{Pr(x_i = 0 \mid f(\vec{x}) = 1)} \geq \log \theta \qquad \text{if and only if}$$

$$\sum_i a_i \xi_i \geq T,$$

where

$$a_i = \log \frac{Pr(x_i = 1 \mid f(\vec{x}) = 1)}{Pr(x_i = 0 \mid f(\vec{x}) = 1)}$$

and $T = \log \theta$. T can be calculated from the various probabilities that we know, as are the a_i, but the general philosophy here is to generate the a_i as specified, and to pick the T so as to optimize performance.

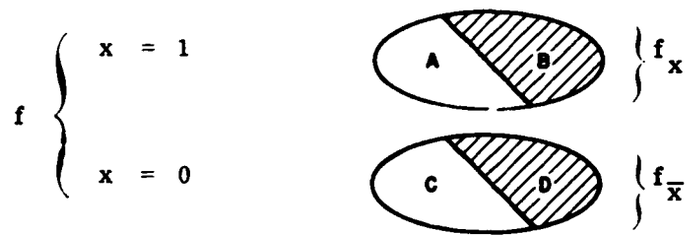
---

* M. E. Maron, "Design Principles for an Intelligent Machine", IRE Trans. Inf. Theory, Vol. IT-8, pp. 179-185, September 1962.

We will now discuss the following question: If we intend to use a threshold function for the recognition procedure, does the above-stated point of view provide the best, or even good, sets of weights a_i?

Saul Amarel (RCA Laboratories) has suggested that the independence assumption is a very severe one, and that only a small proportion of threshold functions might actually be realized by the above procedure. This observation has motivated the investigations reported below — indeed, using the same data, a simpler rule will be shown to work better, and allow a startling improvement; a nonprobabilistic switching theory viewpoint is applied to obtain these results.

We facilitate a comparison by considering the following situation: All 2^n of the $\vec{\xi}$ are given equal probability of occurring, and each $f(\vec{\xi})$ is specified. We assume that f is a threshold function. Probabilities are obtained simply by counting frequencies: Let f_{x_i} be the restriction of f obtained by setting $x_i = 1$, $f_{\bar{x}_i}$ obtained by setting $x_i = 0$, and let m(f) be the number of different $\vec{\xi}$ such that $f(\vec{\xi}) = 1$. Schematically, suppose the 2^n possible $\vec{\xi}$ are mapped as in the diagram ($f(\vec{\xi}) = 1$ in the shaded area, 0 otherwise):



If A, B, C, and D are the number of input vectors $\vec{\xi}$ in the four regions, respectively, then $A = m(f_x)$, $B = m(f_{\bar{x}})$, $C = m(f_{\bar{x}})$, and $D = m(f_{\bar{x}})$. Note that $m(f) = B + D = m(f_x) + m(f_{\bar{x}})$, $m(\bar{f}_x) + m(f_x) = 2^{n-1}$, etc. Since the $\vec{\xi}$ are equiprobable, we have

$$p_i = \frac{m(f_{x_i})}{m(f)}$$

$$r_i = \frac{m(\bar{f}_{x_i})}{m(\bar{f})}$$

$$p_o = \frac{m(f)}{2^n} .$$

Theorem: The independence assumptions hold only for constant functions and functions which are the identity (or complement) in some argument.

Proof: If f is constant (0 or 1), the independence is trivial; so suppose f is not a constant function. By simple properties of threshold functions, we now know that we can, without loss of generality, assume that $f(\vec{0}) = 0$ and $f(1) = 1$, where

$$\vec{0} = (0, 0, \ldots, 0)$$
$$1 = (1, 1, \ldots, 1) .$$

Thus

$$\Pr(\vec{x} = \vec{1} \mid f(\vec{x}) = 0) = 0,$$

so that if the independence assumptions hold,

$$0 = \prod_i r_i = \prod_i \frac{m(\bar{f}_{x_i})}{m(\bar{f})} .$$

But one of the factors must be zero, so for that i,

$$\begin{aligned} m(f) &= 2^n - m(\bar{f}) = 2^n - [\, m(\bar{f}_{x_i}) + m(\bar{f}_{\bar{x}_i}) \,] \\ &= 2^n - m(\bar{f}_{\bar{x}_i}) \geq 2^n - 2^{n-1} = 2^{n-1} . \end{aligned} \tag{1}$$

On the other hand

$$\Pr(\vec{x} = \vec{0} \mid f(\vec{x}) = 1) = 0.$$

So, by the independence assumption

$$0 = \prod_i (1 - p_i) = \prod_i \left(1 - \frac{m(f_{x_i})}{m(f)}\right) .$$

Again some factor, say the j^{th}, must be zero:

$$m(f) = m(f_{x_j}) . \tag{2}$$

But now, from Eqs. (1) and (2),

$$2^{n-1} \leq m(f) = m(f_{x_j}) \leq 2^{n-1},$$

so that all the terms are equal. Thus for some j, f_{x_j} must be the constant 1. Also, since $m(f_{\bar{x}_j}) = m(f) - m(f_{x_j}) = 0$, $f_{\bar{x}_j}$ must be the constant 0. Thus $f(x) = x_j$.
(END OF PROOF.)

Thus we have shown that for any interesting threshold function, with equiprobable input combinations, the independence assumptions in fact do not hold. The theorem can probably be extended to more general input distributions, - possibly to arbitrary distributions. This substantiates S. Amarel's conjecture, and suggests that the Bayesian approach can be improved. In the next section we consider a different viewpoint.

B. GEOMETRIC SYNTHESIS

A standard switching-theoretic geometric interpretation of threshold functions is as follows: The inputs take on values ±1, the input n-tuples ξ are mapped correspondingly onto the 2^n corners of a 2 x 2 x ... x 2 cube centered on the origin of a Euclidean n-space. For a given f, those corners (or vertices) for which $f(\xi) = 1$, are called 1-vertices; if $f(\xi) = 0$, the vertex is a 0-vertex. An (n + 1)-tuple $a_0, a_1, \ldots, a_n$ of real numbers is a realization of f when the hyperplane

$$a_0 + a_1 x_1 + \ldots + a_n x_n = 0$$

separates the cube so that 0-vertices lie on one side, 1-vertices on the other; when such a hyperplane exists, f is a threshold function. (Note: We still use 0 and 1 for outputs.)

In order that we can treat the threshold in the same manner as other weights, and so that the hyperplane will pass symmetrically through the origin, we add a new dimension and variable x_0; when $x_0 = +1$ we define the new function to agree with our original function; when $x_0 = -1$, we define the new function to agree with the dual of the original. (This is the self-dualization idea of [Goto-Takahasi].)

Now: An appropriate normal form for the equation of such a hyperplane is

$$x_0 \cos \alpha_0 + x_1 \cos \alpha_1 + x_2 \cos \alpha_2 + \ldots + x_n \cos \alpha_n = 0,$$

where the α_i are direction angles between a normal to the plane and the corresponding coordinate axes. We will relate direction angles to area on an n-sphere, and then area on the n-sphere to the $m(f_{x_i})$, by straightforward geometric arguments, described and sketched in Fig. 1.

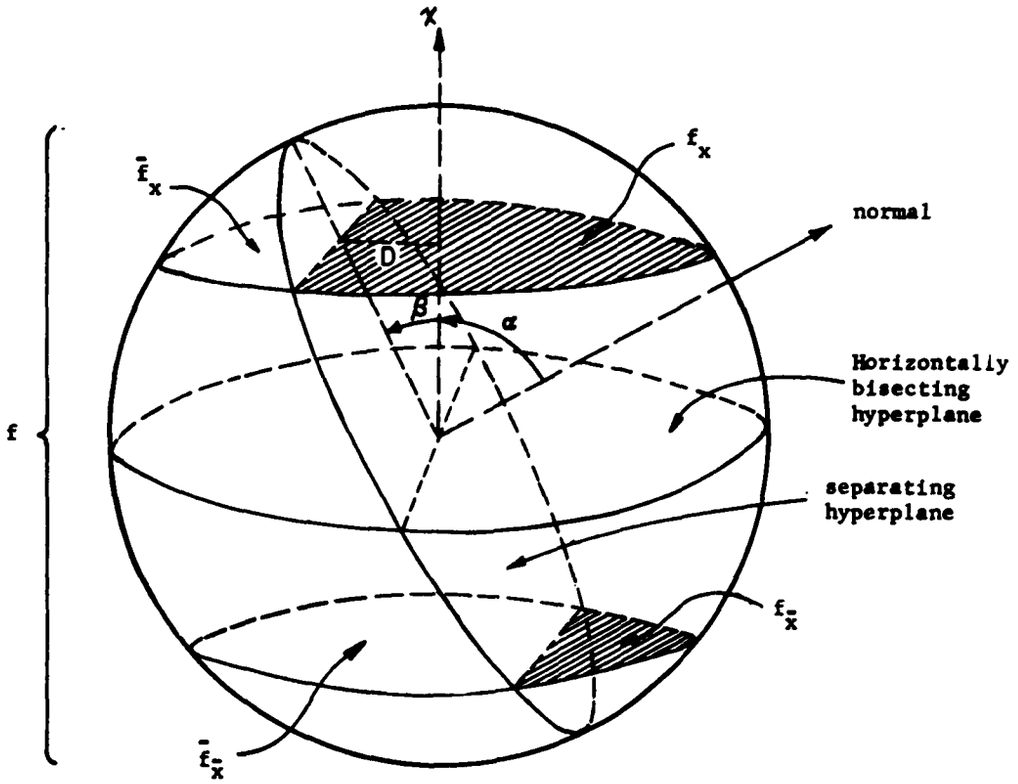


Fig. 1 Sketch of n-sphere

Suppose we pass an origin-centered (n+1)-sphere of radius $\sqrt{(n+1)}$ through the 2^{n+1} vertices of the "cube" which we have defined in E^{n+1}. For concreteness, let us consider now a particular coordinate (say x) which we consider "up". (We should speak of an x_i; "i" is dropped for convenience. We want to find a = cos α for this x.) All the other axes define a hyperplane, which we call "horizontal". If we pass hyperplanes parallel to this one, at a distance 1 "above" and 1 "below", we have isolated the regions where x = +1 and x = -1, respectively; this is where f_x and $f_{\bar{x}}$ are represented. We will consider only f_x below; by our construction, $f_{\bar{x}}$ is just its dual. Within the n-dimensional region containing f_x, the original (n+1)-sphere defines an n-sphere by intersection. Furthermore, let us imagine our separating hyperplane passed through the figure; it will divide the n-sphere into two parts (corresponding to 0-vertices

and 1-vertices). Now, since the 2^n vertices are spread evenly over the surface of the n-sphere, the ratio $m(\bar{f}_x)/m(f_x)$ is an estimate of the ratio of areas cutoff by the separating hyperplane. For greater convenience, we define the ratio

$$r = \frac{(\text{Area on 1-vertex side in upper n-sphere}) \ -1/2 \ (\text{Area of n-sphere})}{1/2 \ (\text{Area of n-sphere})}$$

$$\approx \frac{m(f_x) \ -2^{n-1}}{2^{n-1}}$$

(r will have the same sign as the final weight; r = 0 when f is insensitive to x.)

We assume that r is known by counting (or sampling) vertices; we see next that this ratio determines the distance, D, which separates the separating hyperplane (as induced in the upper n-sphere) from the origin (of the upper n-sphere). With a bit of trigonometry, it can be seen that

$$dr = \frac{2\omega_{n-1} \ (\sqrt{n-D^2})^{n-2}}{\omega_n \ (\sqrt{n})^{n-1}} \frac{\sqrt{n}\, dD}{\sqrt{n-D^2}}$$

(where ω_n is the area of the unit n-sphere), so that

$$\frac{dD}{dr} = \frac{\sqrt{n}\ \omega_n}{2\omega_{n-1}} \quad (1 - \frac{D^2}{n})^{\frac{3-n}{2}} .$$

Set

$$k = \frac{\sqrt{n}\ \omega_n}{2\ \omega_{n-1}} .$$

Initial conditions are that r = 0 when D = 0 (and r = 1 when $D = \sqrt{n}$).

But if we know D (in terms of r), then we can determine the angle β between the separating hyperplane in (n+1)-space and the x-axis — we have a right triangle with adjacent side 1, opposite side D. The direction angle α is the complement of β, so that

$$a = \cos\alpha = \sin\beta = \frac{D}{\sqrt{1+D^2}} = \frac{\text{sign (D)}}{\left(\frac{1}{D^2}+1\right)^{1/2}} .$$

(We drop the "sign (D)", but in practice it is used, of course, to determine the sign!) Solving for D:

$$D^2 = \frac{a^2}{1-a^2} .$$

Now we can calculate

$$\frac{da}{dr} = \frac{da}{dD}\frac{dD}{dr} = (D^2+1)^{-3/2} \cdot k\left(1 - \frac{D^2}{n}\right)^{\frac{3-n}{2}}$$

$$= k(1-a^2)^{n/2} \left(1 - \frac{n+1}{n} a^2\right)^{\frac{3-n}{2}}$$

(after substituting and rearranging). Noting that $a = r = 0$ at $D = 0$, we calculate an expansion for a around $r = 0$:

$$a = kr - k^3\left(2 + \frac{3}{n}\right)\frac{r^3}{6} + k^5\left(22 + \frac{54}{n} + \frac{45}{n^2}\right)\frac{r^5}{120} - \ldots$$

The constant

$$k = \frac{\sqrt{n}\ \omega_n}{2\ \omega_{n-1}} = \frac{\sqrt{n}\,\frac{2\pi^{n/2}}{\Gamma\left(\frac{n}{2}\right)}}{2 \cdot \frac{2\pi^{\frac{n-1}{2}}}{\Gamma\left(\frac{n-1}{2}\right)}} = \frac{\sqrt{n\pi}\,\Gamma\left(\frac{n-1}{2}\right)}{2\,\Gamma\left(\frac{n}{2}\right)} ,$$

$k \to \sqrt{\frac{\pi}{2}}$ as $n \to \infty$ (Wallis' Theorem).

If we take out the dummy x_o and divide all weights by $k \cdot 2^{-n}$ and set $k^{2(2+3/n)/6} \cdot \lambda$, we obtain

$$a_i = \left[m(f_{x_i}) - m(f_{\bar{x}_i}) \right] - \lambda \left[m(f_{x_i}) - m(f_{\bar{x}_i}) \right]^3 + \ldots$$

and threshold

$$T' = \left[2^{n-1} - m(f) \right] - \lambda \left[2^{n-1} - m(f) \right]^3 + \ldots$$

This threshold is for a ± 1 system; using a standard rule, the threshold for a (0, 1) system (as defined in the introduction) is then

$$T = 1/2\,(T' + a_1 + a_2 + \ldots + a_n) .$$

Experience in the application of these rules has been very good: Often the first approximation, $a_i = m(f_{x_i}) - m(f_{\bar{x}_i})$ and $T' = 2^{n-1} - m(f)$. realizes given threshold functions. For all functions tried (including a complicated ten-argument example), it was easy to find a λ so that the second approximations worked; the values were very close to the predicted λ's. Randomized experiments on a computer are planned to check the general worth of the second approximation, to determine the variation of λ from function to function for a given n, and to determine a good value of λ as a function of n. ($\lambda = 0.0008$ was a good value for $n = 7$, $\lambda = 0.00001$ for $n = 10$.)

An important note: Chow [Chow-2] has shown, along with other theoretically very interesting results, that the signs and relative magnitudes of the a_i assigned by the first approximation $a_i = m(f_{x_i}) - m(f_{\bar{x}_i})$ are always correct.

C. COMPARISON AND COMMENTS

Theorem: The Bayes approach produces a set of weights worse than does the first geometric approximation, described above.

Proof: Assume that the weights should all be positive; if they are not, suitable complementations make them so. It is clear from experience with the first approximation, and follows from the fact that a better approximation subtracts a cubic term from the first, that the first approximation gives exaggeratedly high first estimates to the larger a_i; in other words, for $a_i > a_j$, the ratio

$$\frac{m(f_{x_i}) - m(f_{\bar{x}_i})}{m(f_{x_j}) - m(f_{\bar{x}_j})}$$

is too high (for the best chance of realization). Let us abbreviate, putting $m_i = m(f_{x_i})$, and $m = m(f)$. Now we will show that the Bayesian weights

$$a_i = \log \frac{m_{i/m}}{1 - m_{i/m}} = \log \frac{m_i}{m - m_i}$$

are worse yet, because for $m_i > m_j$,

$$\frac{\log \dfrac{m_i}{m - m_i}}{\log \dfrac{m_j}{m - m_j}} > \frac{m_i - (m - m_i)}{m_j - (m - m_j)} = \frac{m_i - m/2}{m_j - m/2}$$

which is already too large. We know that $0 \leq m_j < m_i \leq m$, and we drop the equality possibilities, where the Bayes weights blow up. Furthermore, because of the positivity assumption (and [Chow-2], for example), $m(f_{x_j}) > m(f_{x_i})$, i.e. $m_j > m - m_j$, so $m_j > m/2$. Putting $y = \frac{m_i}{m}$ and $x = \frac{m_j}{m}$, we want then to prove: For $1/2 < x < y < 1$,

$$\frac{\log \frac{y}{1-y}}{\log \frac{x}{1-x}} > \frac{y - 1/2}{x - 1/2} .$$

Since $\log \frac{x}{1-x} > 0$ and $x - 1/2 > 0$, we need only show that the function

$$g(x,y) = (x-1/2) \log \frac{y}{1-y} - (y-1/2) \log \frac{x}{1-x}$$

is positive in the range R: $1/2 < x < y < 1$. But

$$\frac{\partial g}{\partial y} = (x-1/2) \cdot \frac{1-y}{y} \cdot \left[\frac{1}{1-y} + \frac{y}{(1-y)^2}\right] - \log \frac{x}{1-x}$$

$$= \frac{(x-1/2)}{y(1-y)} - \log \frac{x}{1-x}$$

and

$$\frac{\partial}{\partial x} \frac{\partial g}{\partial y} = \frac{1}{y(1-y)} - \frac{1}{x(1-x)} = \frac{(y-x)(y+x-1)}{y(1-y)x(1-x)}$$

which is positive in R. Furthermore

$$\left.\frac{\partial g}{\partial y}\right]_{x=1/2} = 0,$$

(the left boundary of R), so that $\frac{\partial g}{\partial y}$ is positive in R. (There are no discontinuities.) But

$$g(x,x) = 0$$

(the lower boundary of R), so that g is positive in R (again no discontinuities), as we wanted to show. Thus the ratio of larger weights to smaller weights is greater in the Bayesian approach than in the first geometric approximation, where it is already too high. (END OF PROOF.)

We have shown that the Bayesian approach is poor for the synthesis of threshold functions, assuming equiprobable inputs and complete specification. It should be clear that neither of these are important in geometric synthesis, however: Probabilities of inputs play no role, and the areas can be estimated from samples, when the function is incompletely specified. (Use a weighting density function so that closely packed sample points have their proper effect relative to sparsely packed ones.) Whether the Bayesian approach is inferior to the geometric in the case of nonthreshold functions, has not been investigated. (Here probabilities mean something — the problem is to minimize the probability of error, say, when a threshold function is used as an approximation.)

As an example of these ideas, consider the function expressed by

$$A\ [\ B(C+D+E)+CD(E+F)+(C+D)\ EF\ ]\ +BCD(E+F),$$

which has (minimum integral) realization

7, 6, 4, 4, 3, 2, threshold 16.

The weights generated by the Bayesian approach are

$$\log \frac{19}{3}\ ,\ \log \frac{17}{5}\ ,\ \log \frac{15}{7}\ ,\ \log \frac{15}{7}\ ,\ \log \frac{14}{8}\ ,\ \log \frac{13}{9}$$

or (dividing through by .08)

10.04, 6.66, 4.15, 4.15, 3.04, 2.00.

With these weights, at best 62 out of the 64 vertices can be separated properly (threshold 19.2). Note the characteristic manner in which higher weights are exaggerated, as compared with the minimal realization.

The first geometric approximation is

19-3, 17-5, 15-7, 15-7, 14-8, 13-9, threshold 32

or (dividing by 2)

8, 6, 4, 4, 3, 2, threshold 16.

This realization also fails on two vertices; if optimization by varying the threshold is allowed, the same weights, with threshold 17, fails only on one vertex. Any (positive) λ up to 0.004 will, in the second geometric approximation, give a realization of the original function; the theoretically calculated value of 0.0034 works fine.

A final comment: The iterative synthesis of Mattson* apparently begins with our first geometric approximation as initial trial (although no rigorous basis is suggested). Mattson's idea is to optimize on one weight (or the threshold) at a time. As with the Bayesian approach, however, threshold functions will usually not be realized. J. Sklansky of RCA Laboratories raised this issue by constructing a "pathological" plane which almost realizes $A(B+C)$, but which cannot be transformed to realization by changing any one coefficient. The example discussed above was subsequently chosen to illustrate that even with a very good initial try (in fact, the try which I believe Mattson intended), namely

$$8,\ 6,\ 4,\ 4,\ 3,\ 2,\ \text{threshold } 6$$

(in the ± 1 system), which misses only the 1-vertex

$$\bar{A}BCD\bar{E}F:\ -8+6+4+4-3+2 = 5 < 6,$$

Mattson's iteration won't converge: If a change in some weight is to improve the realization, it will clearly have to be a decrease in the weight a of A, an increase in b or c or d, a decrease in e, an increase in f, or a decrease in T. But a decrease in a loses the 1-vertices $AB\bar{C}\bar{D}E\bar{F}$, $A\bar{B}\bar{C}\bar{D}EF$, and $A\bar{B}\bar{C}DEF$ at the same time it gains $\bar{A}BCD\bar{E}F$ — this is no progress. Similarly increasing b loses 1-vertex $A\bar{B}\bar{C}\bar{D}EF$ and $A\bar{B}\bar{C}DEF$ and "gains" 0-vertex $AB\bar{C}\bar{D}\bar{E}F$; increasing c loses $AB\bar{C}\bar{D}E\bar{F}$ and "gains" $A\bar{B}CD\bar{E}\bar{F}$; d symmetrically; decreasing e loses $AB\bar{C}\bar{D}E\bar{F}$, $A\bar{B}C\bar{D}EF$, and $A\bar{B}\bar{C}DEF$; increasing f loses $AB\bar{C}\bar{D}E\bar{F}$ and "gains" $A\bar{B}CD\bar{E}\bar{F}$; decreasing T "gains" $AB\bar{C}\bar{D}\bar{E}F$ and $A\bar{B}CD\bar{E}\bar{F}$. Thus a combination of changes will be needed to effect a realization by iteration; the "hill" being climbed requires a more subtle approach. This is strong reason for considering more direct methods, as is outlined above.

The main conclusion of this section is: If we decide in a recognition problem to try to use a threshold decision function, then switching theoretic methods for synthesizing the function are preferable to the conventional Bayesian approach.

---

* R. L. Mattson, " A Self-Organizing Binary System, " PROC EJCC, pp. 212-217, December 1959.

D. CALCULATION OF MEASURES

The calculation of the quantities $m(f)$, $m(f_{x_i})$ from an algebraic representation of f can be made very easily, providing f is 2-monotonic, as follows:

1. Put the representation into canonical form (page 67 of thesis [W-4]); call this form P. (We will assume P is positive, but nearly the same procedures apply in the general case.)

2. Write the numbers 2^{n-i} (decimal form) under the variables x_i, in each appearance as the last factor of a term of P. Then $m(f)$ is simply the sum of these numbers.

Example: (x_1 is A, etc.)

$$A[\,B(C + D + EF) + C(D(E + F) + EF)\,] + BCDE$$

$$8 \quad 4 \quad 1 \qquad\qquad 2 \quad 1 \quad 1 \qquad\qquad 2$$

$$m(f) = 8 + 4 + 1 + 2 + 1 + 1 + 2 = 19.$$

Proof: In general,

$$m(f) = m(f_x) + m(f_{\bar{x}})$$

for any argument x. But f is 2-monotonic by assumption, and P, being in canonical form, is

$$P = x_1 Q + R$$

where Q and R don't involve x_1. Thus

$$m(f) = m(g + h) + m(h)$$

where g is represented by Q, h by R. But by theorem 18 (page 67) of thesis [W-4], $R \rightarrow Q$, so that $h \rightarrow g$; i.e., $g + h = g$. Then

$$m(f) = m(g) + m(h).$$

This simple rule of decomposition, applied recursively, gives the rule stated above.

(END OF PROOF)

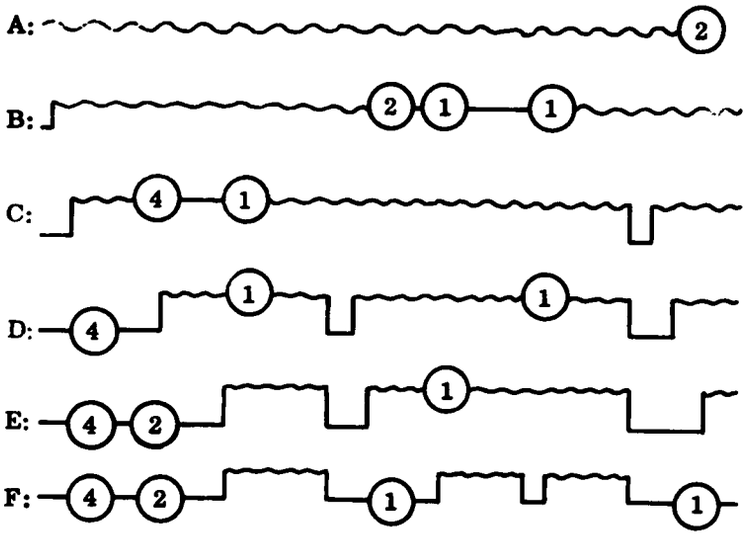
3. To compute the m (f_{x_i}), below each numerical entry of the previous step, except the 1's, write one half of that entry.

4. Compute $m(f_{\bar{x}_i})$ as follows: Moving from the left and starting on the bottom row, add up entries until an x_i is encountered; then skip over whatever the x_i might multiply into, and move up to the upper row, adding entries until an x_j with $j \leq i$ is encountered. Then move back down again, and continue as before. The sum will be $m(f_{\bar{x}_i})$; then $m(f_{x_i}) = m(f) - m(f_{\bar{x}_i})$. (It's easy to generate a slightly more complicated rule to calculate $m(f_{x_i})$ directly; this can be used as a check.)

Example:

$$A[B(C + D + EF) + C(D(E + F) + EF)] + BCDE$$

8	4	1		2	1	1	2
4	2			1			1



2. 19 - 2 = 17.

4. 19 - 4 = 15.

5. 19 - 5 = 14.

6. 19 - 6 = 13.

7. 19 - 7 = 12.

8. 19 - 8 = 11.

VII. REFERENCES

Only English-language papers commercially available are listed below. Almost all papers on threshold logic not covered by them are Lockheed (Missiles and Space Division) Technical Notes and Reports, authored by D. T. Perkins, D. G. Willis, E. A. Whitmore, and Sze-Tsen Hu. Also, at a Lockheed Symposium in February 1962, Willis gave a paper "Minimum Weights for Threshold Switches", and E. Goto a paper "Threshold, Majority and Bilateral Switching Devices" -- the latter seems to be subsumed by [Goto-Takahasi]. Papers in Italian, Japanese, and Russian do not appear to contain material not covered by the references below.

The "[R62-33]," etc., give the number of a review of the cited paper, in the IRE Transactions on Electronic Computers. "SCTLD" are the Proceedings of the Annual Symposia on Switching Circuit Theory and Logical Design; the 1960 and 1961 proceedings are Publication S-134 of the AIEE, the 1962 proceedings are Publication S-141.

S. B. Akers [1] "Threshold Logic and Two-Person, Zero-Sum Games, "SCTLD (1961) pp. 27-33, September 1961.
[2] "On the Algebraic Manipulation of Majority Logic," IRE Transactions on Electronic Computers, Vol. EC-10, No. 4, p. 779, December 1961.
[3] "Synthesis of Combinational Logic Using Three-Input Majority Gates" SCTLD (1962), pp. 149-157, September 1962.

S. H. Cameron, "An Estimate of the Complexity Requisite in a Universal Decision Network," Bionics Symposium, WADD Report 60-600, pp. 197-212, December 1960.

C. K. Chow, [1] "Boolean Functions Realizable with Single Threshold Devices," Proc. IRE, Vol. 39, pp. 370-371, January 1961. [R62-95]
[2] "On the Characterization of Threshold Functions," SCTLD (1961) pp. 34-38, September 1961. [R62-33].

C. L. Coates, R. B. Kirchner, P. M. Lewis II, "A Simplified Procedure for the Realization of Linearly Separable Switching Functions", IRE Transactions on Electronic Computers, Vol. EC-11, pp. 447-458, August 1962.

C. L. Coates and P. M. Lewis II, "Linearly Separable Switching Functions," Journal of the Franklin Institute, Vol. 272, pp. 360-410, November 1961 [R62-120]

M. Cohn and R. Lindaman, "Axiomatic Majority-Decision Logic," IRE Transactions on Electronic Computers, Vol. EC-10, pp. 17-21, March 1961. (See also correspondence in IRETEC of September 1961, pg. 530.) [R61-85]

L. Dadda, [1] "Synthesis of Threshold Logic Combinatorial Networks," Alta Frequenza, n.3, Vol. XXX, pp. 224-28E-35E-231, 1961. [R61-86]
[2] "Synthesis of Threshold Switching Networks by Map Methods," (Abstract) Proceedings of the IFIP Congress 62, to be published by North Holland. Conference: Munich, August 1962.

S. N. Einhorn, "The Use of the Simplex Algorithm in the Mechanization of Boolean Switching Functions by Means of Magnetic Cores," IRE Transactions on Electronic Computers, Vol. EC-10, No. 4, pp. 615-622, December 1961. (See also correspondence in IRETEC of August 1962, pg. 573).

C. C. Elgot, "Truth Functions Realizable by Single Threshold Organs," SCTLD (1960), pp. 225-245, September 1961, and AIEE Conference Paper 60-1311, October 1960, revised November 1960.

C. C. Elgot, S. Muroga, "Two Open Problems on Threshold Logic," SCTLD (1961), pg. 166, September 1961.

P. Ercoli, L. Mercurio, "Threshold Logic with one or more than one Threshold," Proceedings of the IFIP Congress 62, to be published by North Holland. Conference: Munich, August 1962.

I. J. Gabelman, [1] "The Functional Behavior of Majority (Threshold) Elements", Doctoral Dissertation for Syracuse University, Electrical Engineering Department. Available through University Microfilms. June 1961.

[2] "A Note on the Realization of Boolean Functions Using a Single Threshold Element," Proc IRE, Vol. 50, No. 2, pp. 225-226, February 1962. [R62-95]

[3] "The Synthesis of Boolean Functions Using a Single Threshold Element", IRE Tranactions on Electronic Computers, Vol EC-11, No. 5, pp. 639-642, October 1962.

E. Goto, H. Takahasi, "Some Theorems Useful in Threshold Logic for Enumerating Boolean Functions," Proceedings of the IFIP Congress 62, to be published by North Holland. Conference: Munich, August 1962.

W. H. Highleyman, "A Note on Linear Separation," IRE Transactions on Electronic Computers, Vol. EC-10, No. 4, pp. 777-778, December 1961. [R62-95]

G. Hotz, "Digital Filters of Threshold Elements," Proceedings of the IFIP Congress 62, to be published by North Holland. Conference: Munich, August 1962.

W. H. Kautz, "The Realization of Symmetric Switching Functions with Linear-Input Logical Elements," IRE Transactions on Electronic Computers, Vol. EC-10, No. 3, pp. 371-378, September 1961 [R62-2]

P. M. Lewis II, C. L. Coates, "A Realization Procedure for Threshold Gate Networks," SCTLD (1962), pp. 159-168, September 1962.

R. Lindaman, [1] "A New Concept in Computing," Proc. IRE, Vol. 48, p. 257, February 1960.

[2] "A Theorem for Deriving Majority-Logic Networks within an Augmented Boolean Algebra," IRE Transactions on Electronic Computers, Vol. EC-9, pp. 338-342, September 1960.

R. McNaughton, "Unate Truth Functions," IRE Transactions on Electronic Computers, Vol. EC-10, pp. 1-6, March 1961, and previously available as Technical Report No. 4, Applied Mathematics and Statistics Laboratory, Stanford University, October 1957. [R61-85]

S. Muroga, [1] "Logical Elements on Majority Decision Principle and Complexity of Their Circuits", Proceedings of the International Conference on Information Processing (June 1959), Columbia Univ. Press 1960.

S. Muroga, I. Toda, and S. Takasu, [2] "Theory of Majority Decision Elements," Journal of the Franklin Institute, Vol. 271, pp. 376-418, May 1961. [R62-119]

[3] "Functional Forms of Majority Functions and a Necessary and Sufficient Condition for their Realizability," SCTLD (1961), pp. 39-46, September 1961. [R62-35]

[4] "Restrictions in Synthesis of a Network with Majority Elements," Proc. IRE, Vol. 49, p. 1455, September 1961.

[5] "Majority Logic and Problems of Probabilistic Behavior," Self Organizing Systems 1962, pp. 243-281, Spartan Books, 1962, (Proceedings of Conference May 1962).

[6] "Generation of Self-Dual Threshold Functions and Lower Bounds of the Number of Threshold Functions and a Maximum Weight," SCTLD (1962), pp. 169-184, September 1962.

S. Muroga, I. Toda, M. Kondo, [7] "Majority Decision Functions of up to Six Variables", to appear in the Journal of Math. of Comp., October 1962, (Published in Japanese in 1959 and 1960).

H. S. Miiller and R. O. Winder, "Majority Logic Synthesis by Geometric Methods," IRE Transactions on Electronic Computers, Vol. EC-11, pp. 89-90, February 1962. [Also, Scientific Report No. 4 on this Contract.]

R. C. Minnick, "Linear-Input Logic," IRE Trnasactions on Electronic Computers, Vol. EC-10, pp. 6-16, March 1961; essential ideas delivered at the sixth Annual Symposium on Computers and Data Processing of the Denver Research Institute, July 1959. [R61-85]

J. Myhill and W. H. Kautz, "On the Size of Weights Required for Linear-Input Switching Functions," IRE Transactions on Electronic Computers, Vol. EC-10, pp. 288-290, June 1961 [R62-96]

M. C. Paull and E. J. McCluskey, Jr., "Boolean Functions Realizable with Single Threshold Devices," Proc. IRE, Vol. 48, pp. 1335-1337, July 1960. [R62-95]

R. C. Singleton, "A Test for Linear Separability as Applied to Self-Organizing Machines," Self Organizing Systems 1962, pp. 503-524, Spartan Books, 1962 (Proceedings of the Conference, May 1962).

O. B. Stram, [1] "Arbitrary Boolean Functions of N Variables Realizable in Terms of Threshold Devices," Proc. IRE, Vol. 49, pp. 210-220, January 1961. [R61-85] [R62-76]

[2] "The Profile Technique for the Design of Threshold Device Logic," SCTLD (1961), pp. 47-54, [R62-76].

V. I. Varshavskii, [1] "Functional Possibilities and Synthesis of Threshold Elements," Soviet Physics Doklady, Vol. 6, No. 8, pp. 678-680, February 1962. [R62-95]

[2] "On the Complexity of Networks of Depth Two Formulated From Threshold Elements," Soviet Physics Doklady, Vol. 6, No. 8, pp. 683-685, February 1962. [R62-95]

R. O. Winder, [1] "Single Stage Threshold Logic," SCTLD (1960), pp. 321-332, September 1961, and AIEE Conference Paper 60-1261, October 1960.*

[2] "More About Threshold Logic," SCTLD (1961), pp. 55-64, September 1961 [R62-31] [Also, Scientific Report No. 1 on this Contract.]*

[3] "Some Recent Papers in Threshold Logic," Proc. IRE, Vol. 49, p. 1100, June 1961

*This material is incorporated in Special Scientific Report No. 1, on this Contract, or into the present Special Scientific Report.

[4] "Threshold Logic," Doctoral Dissertation for the Mathematics Department of Princeton University. (This paper incorporates the material of [Winder-1] and [Winder-2].) Available through University Microfilms or directly from the author. May 1962.*

[5] "Networks of Threshold Gates," (abstract) Proceedings of the IFIP Congress 62, to be published by North Holland. Conference: Munich, August 1962.*

[6] "Threshold Logic in Artificial Intelligence", Artificial Intelligence (Papers from the AIEE Winter General Meeting, 1962), IEEE Publication S-142, January 1963. [Also Scientific Report No. 6 on this Contract.]*

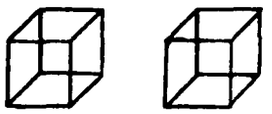
*This material is incorporated in Special Scientific Report No. 1, on this Contract, or into the present Special Scientific Report.

VIII. APPENDIX

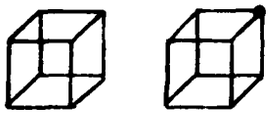
by R. O. Winder

The 83 SD symmetry types for four arguments, with Chow parameters and realizations by 3-input majority gates (optimal number of levels)

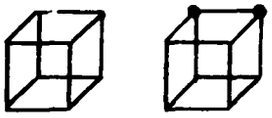
(1) 80000
A



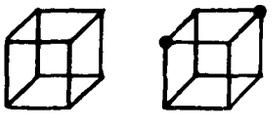
(2) 71111
A(ABC)(ADE)



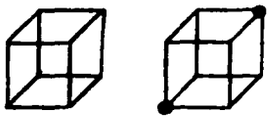
(3) 62220
AB(ACD)



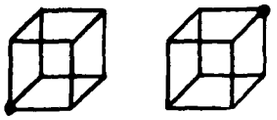
(4) 62200
A(BD̄E)(CDE)



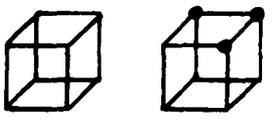
(5) 62000
(ACD)(AC̄Ē)
(BD̄E)



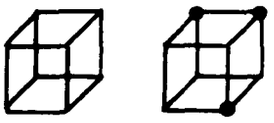
(6) 60000
(ACD)(AB̄Ē)
[(ABD)C̄(ĀBD̄)]



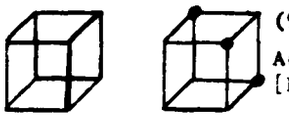
(7) 53311
A(ABC)(ĀDE)



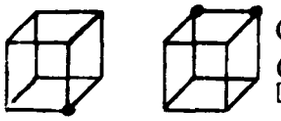
(8) 53111
(ACD)(AC̄E)
(BCD̄)



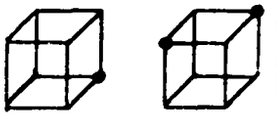
(9) 53111
A(CDE)
[BC̄(BD̄Ē)]



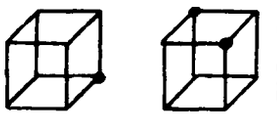
(10) 51111
(ACD)(AB̄C̄)
[BE(ĀBD̄)]



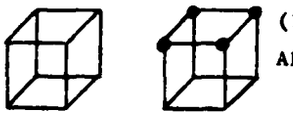
(11) 51111
(ADE)(AB̄Ē)
[(CD̄Ē)(AB̄C̄)(ĀBC̄)]



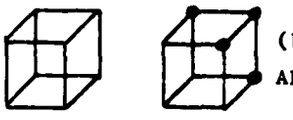
(12) 51111
(AB̄C̄)(CDE)
[(ABĒ)(ABD̄)B]



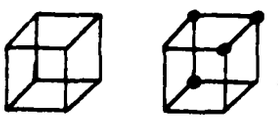
(13) 44400
ABC



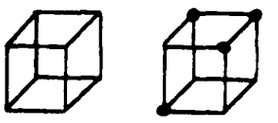
(14) 44222
AB(CDE)



(15) 44220
A(BDE)(BCĒ)

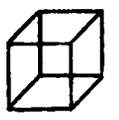


(16) 44200
(ABC)(AC̄D̄)
[(ABD)(BCĒ)(BC̄E)]



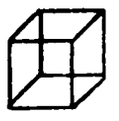
(17) 44000

$A(\bar{A}\bar{C}\bar{D})(BC\bar{D})$



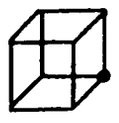
(18) 44000

$(ABD)(AB\bar{D})$
$[(\bar{C}\bar{D}E)(CDE)E]$



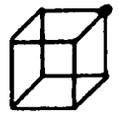
(19) 42222

$(ABC)(A\bar{B}\bar{C})(BDE)$



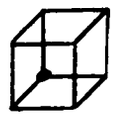
(20) 42222

$(CDE)(B\bar{C}\bar{E})$
$[(ACE)(AB\bar{D})(A\bar{B}D)]$



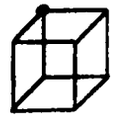
(21) 42220

$(ABC)(\bar{B}CD)$
$[(ABE)(B\bar{C}D)(\bar{A}B\bar{E})]$



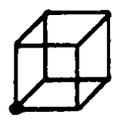
(22) 42220

$(ACE)(BC\bar{E})$
$[(A\bar{C}D)(BD\bar{E})\bar{E}]$



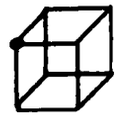
(23) 42200

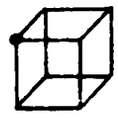
$(ABC)(\bar{C}D\bar{E})$
$[(ACE)(\bar{B}\bar{D}E)(\bar{A}B\bar{E})]$



(24) 42200

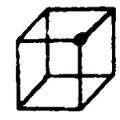
$(ACD)(BC\bar{D})$
$[(A\bar{C}D)(\bar{C}\bar{D}\bar{E})(\bar{A}\bar{B}E)]$





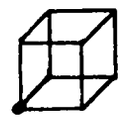
(25) 42200

$(AC\bar{D})(ABD)$
$[(\bar{C}DE)(\bar{A}\bar{D}E)(AC\bar{E})]$



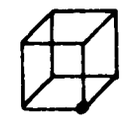
(26) 42200

$[A(ACE)(\bar{A}\bar{B}D)]$
$[(ABC)(ABD)(AB\bar{E})]$
$(\bar{A}\bar{C}\bar{D})$



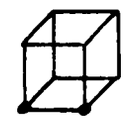
(27) 42000

$(ACD)(A\bar{C}\bar{D})$
$[(ABE)(\bar{A}B\bar{C})(\bar{B}D\bar{E})]$



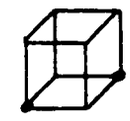
(28) 42000

$(AB\bar{E})(C\bar{D}\bar{E})$
$[(\bar{C}DE)(A\bar{B}\bar{C})(ABD)]$



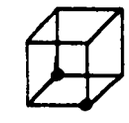
(29) 40000

$(ACD)(\bar{A}B\bar{C})(A\bar{B}\bar{D})$



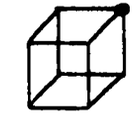
(30) 40000

$(ABC)(A\bar{B}\bar{C})$
$[\bar{A}(A\bar{D}\bar{E})(ADE)]$



(31) 40000

$(ABC)(\bar{B}\bar{D}\bar{E})(\bar{C}DE)$

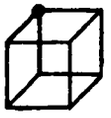


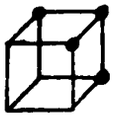
(32) 33333

$(ADE)(BCE)$
$[(BCD)(ABD)E]$

(33) 33331

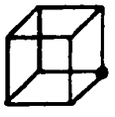
$B(A\bar{B}\bar{E})(CDE)$

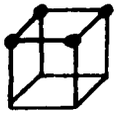




(34) 33311

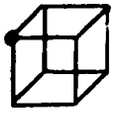
$(ABC)(BDE)$
$[(A\bar{B}\bar{C})(\bar{A}B\bar{C})C]$

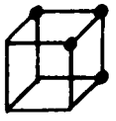




(35) 33311

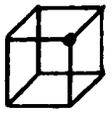
$(AC\bar{D})(ABD)$
$[(CDE)(\bar{A}B\bar{E})(A\bar{B}\bar{E})]$

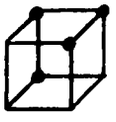




(36) 33311

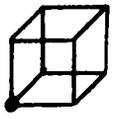
$(ABD)(ACE)(\bar{A}\bar{D}\bar{E})$

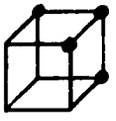




(37) 33111

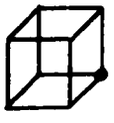
$(C\bar{D}\bar{E})(ABE)$
$[(ABD)(\bar{A}\bar{B}C)\bar{C}]$

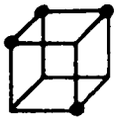




(38) 33111

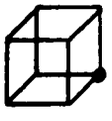
$(ABC)(A\bar{C}E)$
$[(B\bar{D}\bar{E})(CD\bar{E})(\bar{A}\bar{B}\bar{D})]$

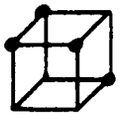




(39) 33111

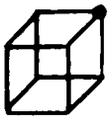
$(ABC)(A\bar{C}D)$
$[(\bar{A}CE)(\bar{C}DE)\bar{E}]$

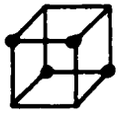




(40) 33111

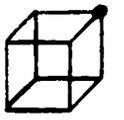
$(AC\bar{D})(BD\bar{E})$
$[(BCD)(B\bar{C}\bar{D})\bar{B}]$

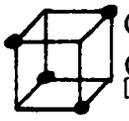




(41) 33111

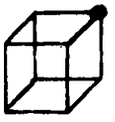
$(CD\bar{E})E$
$[(AB\bar{C})(\bar{C}\bar{D}\bar{E})(AB\bar{E})]$

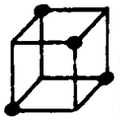




(42) 33111

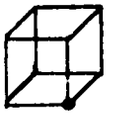
$[B(BDE)(\bar{A}BC)]$
$[A(ACD)(A\bar{B}\bar{E})]$
$[(CDE)(\bar{C}\bar{D}\bar{E})\bar{E}]$

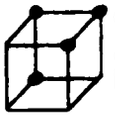




(43) 33111

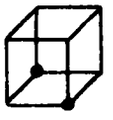
$(ABC)(B\bar{C}E)$
$[(AD\bar{E})(A\bar{B}\bar{D})\bar{A}]$

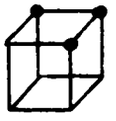




(44) 31111

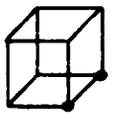
$(AD\bar{E})E$
$[(ABC)(\bar{C}\bar{D}\bar{E})(\bar{A}\bar{B}C)]$

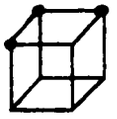




(45) 31111

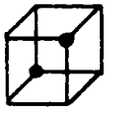
$(ABC)(A\bar{C}E)$
$[(\bar{A}\bar{B}D)(\bar{A}\bar{B}E)C]$

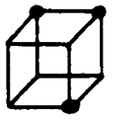




(46) 31111

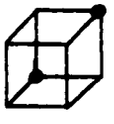
$(CD\bar{E})(A\bar{D}E)$
$[(B\bar{C}D)(\bar{B}\bar{C}E)(BC\bar{D})]$

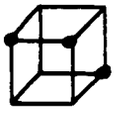




(47) 31111

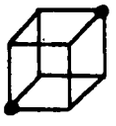
$[(BCE)(ABC)D]$
$[(\bar{B}C\bar{D})(A\bar{B}D)(A\bar{B}\bar{E})]$
$(A\bar{C}D)$

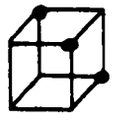




(48) 31111

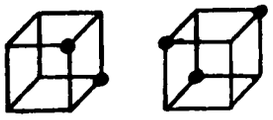
$[A(ACD)(A\bar{B}E)]$
$[(B\bar{D}\bar{E})(B\bar{C}\bar{D})B]$
$(\bar{A}\bar{B}E)$





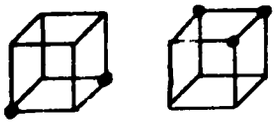
(49) 31111

[(AB̄E)(ACE)D]
[(ABC)(ABD)Ē]
(ĀC̄D̄)



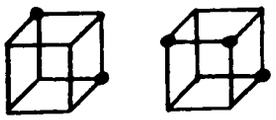
(50) 31111

[(ADE)(ABC)(ĀB̄C)]
[(AB̄C̄)(ĀBC̄)(CDE)]
[(CDE)(ABE)A]



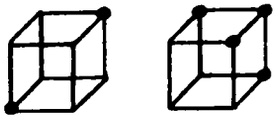
(51) 31111

A[(CDE)(ABC)(ĀBD)]
[(ĀC̄D̄)(ĀC̄Ē)(ĀB̄E)]



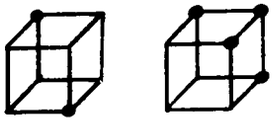
(52) 22222

[(CDE)(ABE)(BDE)]
[A(AD̄Ē)(AB̄C̄)]
[C(ĀBE)(ĀB̄D)]



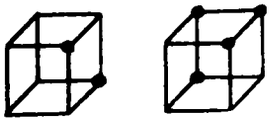
(53) 22222

[(BCD)(CDE)A]
[(ABE)(BD̄E)(AB̄C̄)]
[(B̄C̄E)ĀĒ]



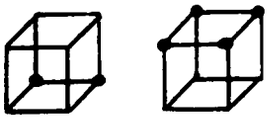
(54) 22222

(ABC)
[(CDE)(ABD)(ĀBE)]
[(B̄C̄E)(B̄C̄D)Ā]



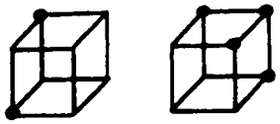
(55) 22220

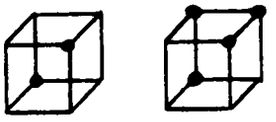
(ABC)(ĀB̄C)
(AC̄D)



(56) 22220

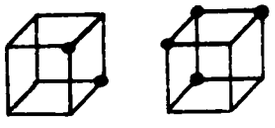
(AB̄Ē)
[A(BCD)(CDE)]
[(ĀBC̄)BD̄]





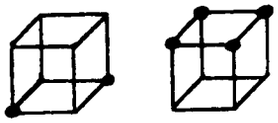
(57) 22220

[(ABD)(CDE)(ĀBD̄)]
[(AD̄Ē)A(ĀBC̄)]
(ABC)



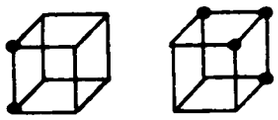
(58) 22220

[(ABC)(B̄CD)(AB̄E)]
[(ABC)(ABD)Ē]
(ĀC̄D̄)



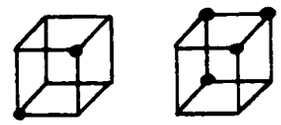
(59) 22200

[(ABC)(BDE)(ĀB̄C)]
[(ABC)Ē(ĀB̄C)]
(C̄D̄E)



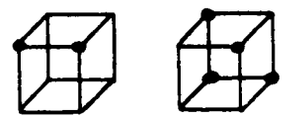
(60) 22200

[Ā(AD̄Ē)(ABC)]
[(ABC)(ABE)(ABD)]
(ĀB̄E)



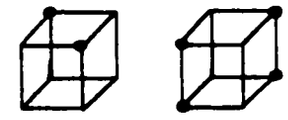
(61) 22200

[Ā (ACE)(ABD)]
[(AB̄Ē)(BD̄Ē)(AB̄C̄)]
[(ĀB̄E)(BCD̄)(AB̄Ē)]



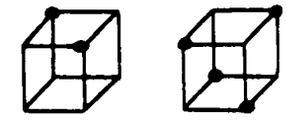
(62) 22200

[(ABD)(ACE)Ā]
[(ACD)(ACE)(AB̄Ē)]
(ĀC̄Ē)



(63) 22200

(AB̄C)(BDĒ)(BD̄E)

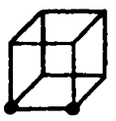


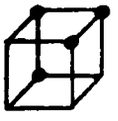
(64) 22200

[(ACD)(AB̄C)E]
[B(BCD)(ABE)]
[(B̄D̄Ē)(ĀB̄C̄)(ĀD̄Ē)]

(65) 22000

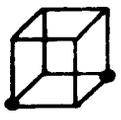
$[(ABC)(ABD)(A\bar{D}E)]$
$[(B\bar{C}\bar{D})(\bar{A}B\bar{C})C]$
$[\bar{B}\bar{E}(\bar{A}\bar{B}D)]$

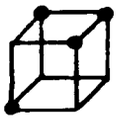




(66) 22000

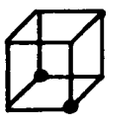
$[(ABC)(CDE)(A\bar{C}\bar{D})]$
$[(\bar{C}D\bar{E})(AB\bar{E})(A\bar{B}C)]$
$(\bar{A}\bar{D}E)$

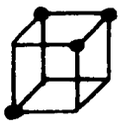




(67) 22000

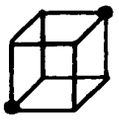
$[(ABC)(ADE)(ABD)]$
$[(\bar{A}B\bar{C})(B\bar{C}E)D]$
$(\bar{B}\bar{D}\bar{E})$

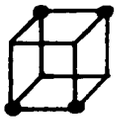




(68) 22000

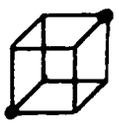
$[(ACD)(ABC)(A\bar{B}E)]$
$[(A\bar{C}\bar{D})(AB\bar{C})(A\bar{B}\bar{E})]$
$\bar{A}$

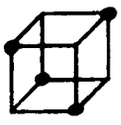




(69) 22000

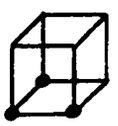
$[A(ACE)(ACD)]$
$[B(\bar{A}B\bar{C})(B\bar{D}\bar{E})]$
$[(\bar{C}\bar{D}\bar{E})(\bar{A}\bar{B}E)(CD\bar{E})]$

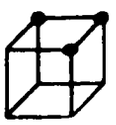




(70) 20000

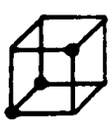
$[(ABD)(ABC)(ACE)]$
$[(\bar{A}B\bar{C})(B\bar{C}\bar{D})(\bar{A}BE)]$
$\bar{B}$

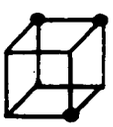




(71) 20000

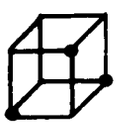
$[A(ABC)(ACD)]$
$[(\bar{B}C\bar{E})(A\bar{B}\bar{C})\bar{D}]$
$[\bar{A}C(\bar{A}B\bar{E})]$

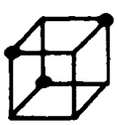


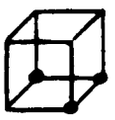


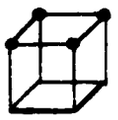
(72) 20000

$[(ACE)(A\bar{B}E)D]$
$[(CD\bar{E})(\bar{A}B\bar{E})(A\bar{B}\bar{C})]$
$[\bar{A}(B\bar{C}\bar{D})(\bar{C}\bar{D}E)]$



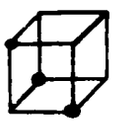


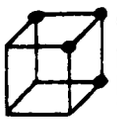




(73) 11111

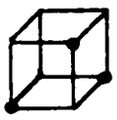
$(ABC)\bar{B}$
$[(\bar{A}B\bar{C})(B\bar{C}D)(\bar{A}BE)]$

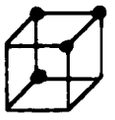




(74) 11111

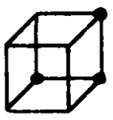
$[(ABC)(ABD)(ABE)]$
$[(\bar{A}B\bar{C})(\bar{A}B\bar{D})(\bar{A}B\bar{E})]$
$[\bar{B}(\bar{B}CD)E]$

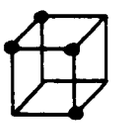




(75) 11111

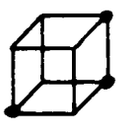
$[(ACE)(BDE)(\bar{A}\bar{B}\bar{E})]$
$[(CD\bar{E})(A\bar{B}\bar{E})(\bar{A}B\bar{C})]$
$(\bar{C}\bar{D}E)$

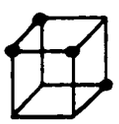




(76) 11111

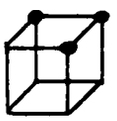
$[(\bar{A}BD)(BDE)(\bar{A}\bar{B}C)]$
$[(A\bar{B}D)(A\bar{C}D)(A\bar{B}E)]$
$[(AB\bar{D})(BC\bar{D})\bar{E}]$

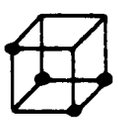




(77) 11111

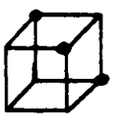
$[\bar{A}(ABC)(ADE)]$
$[A(A\bar{C}\bar{E})(A\bar{B}\bar{D})]$
$[A(\bar{A}\bar{D}E)(\bar{A}\bar{B}C)]$

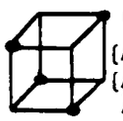




(78) 11111

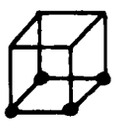
$[(\bar{A}BC)(BCE)D]$
$[(\bar{A}\bar{B}\bar{C})(\bar{B}C\bar{D})(CD\bar{E})]$
$(AB\bar{D})$

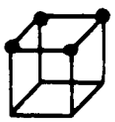




(79) 11111

$\{A\bar{B}[(CDE)(A\bar{D}E)(\bar{C}D\bar{E})]\}$
$\{AB[(CD\bar{E})(\bar{C}\bar{D}\bar{E})E]\}$
$\bar{A}$





(80) 00000

$(ABC)(\bar{A}\bar{B}C)\bar{C}$

(81) 00000

$[(ABC)(ABE)D]$
$[(\bar{A}B\bar{C})(\bar{A}B\bar{E})\bar{D}]$
$\bar{B}$

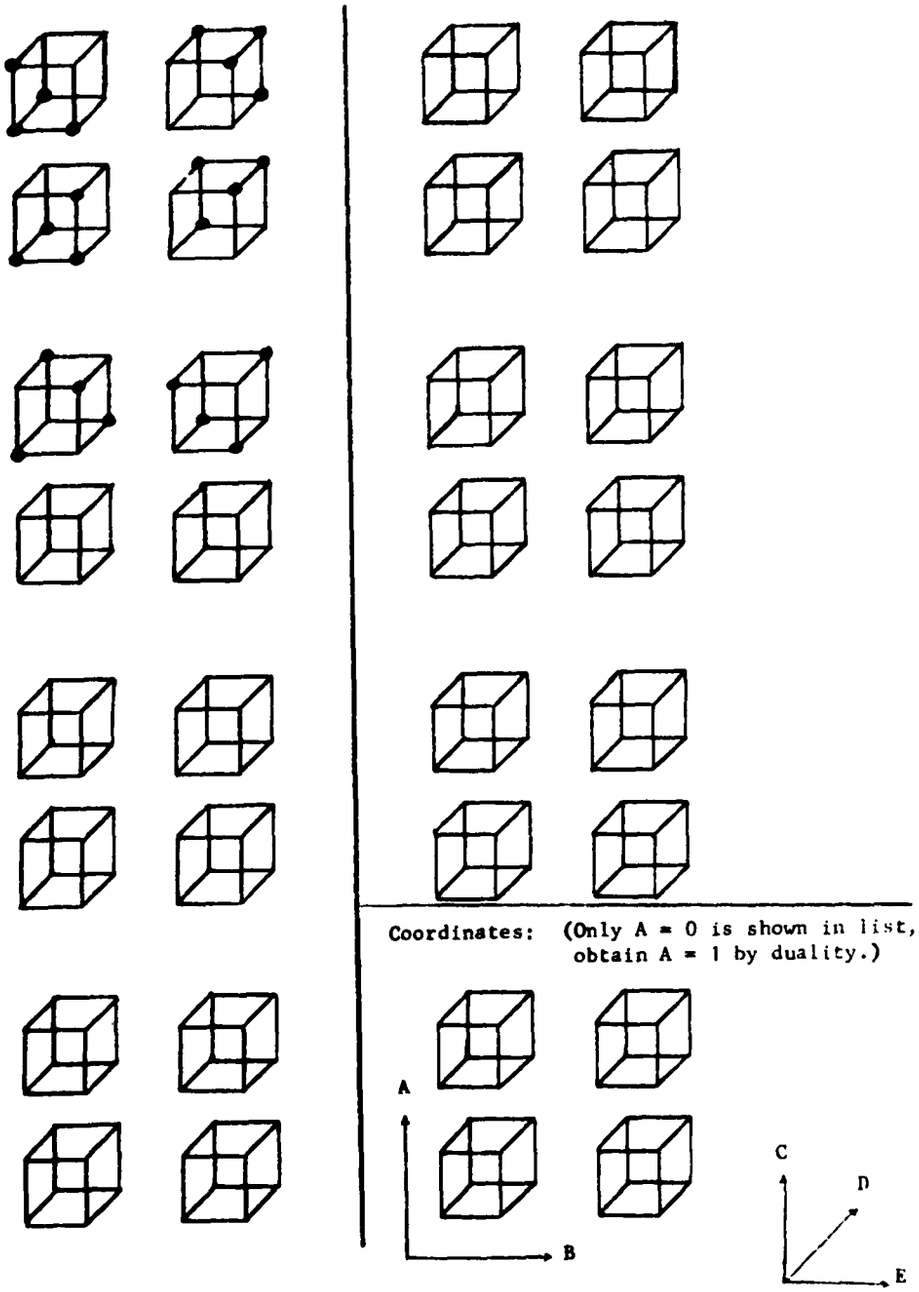
(82) 00000

$[(AC\bar{E})(A\bar{E}D)(\bar{A}\bar{B}C)]$
$[(\bar{A}\bar{D}\bar{E})(\bar{A}\bar{E}D)(ABC)]$
E

(83) 00000

$(DEX)(\bar{D}\bar{E}\bar{X})\bar{E}$

where
$X = (ABC)(\bar{A}\bar{B}C)\bar{C}$



PART 2

RELIABILITY OF SWITCHING NETWORKS

I. INTRODUCTION

This part of Special Scientific Report No. 2 also consists of a series of sections; Section II constitutes a survey of the application of redundancy techniques in switching systems for the improvement of reliability, Section III discusses several extensions of earlier work on a specific scheme (reported in Special Scientific Report No. 1) -- the recursive triangles, and Section IV introduces a new line of exploration: reliability in systems with memory and feedback.

Section II is an outgrowth of the Symposium on Redundancy Techniques for Computing Systems which took place in Washington last year. It is intended to survey the field with the dual purpose of seeing what is being done outside of our work at RCA Laboratories to apply redundancy to increase reliability and evaluating our recursive triangle scheme in the light of what we have observed. We discuss those methods which are most relevant to the problems that we ourselves are considering. (In particular, we discuss the papers presented at the Symposium).

Although many techniques have been suggested for introducing redundancy into computing systems, only those which require very small additional quantities of equipment have gotten beyond the paper design stage. Increased computational capability is a very easy concept to grasp, but the concept of increased reliability is not well understood. In fact,the former is regarded as a goal of the system while the latter is often thought of as an obstacle to be overcome. Consequently,the two have been separated and people have found it easy to justify large additional expense for the former and far more difficult to justify it for the latter. The two are not at all independent but until many more people appreciate the relation between them, the systems proposed to achieve reliability through redundancy will be slow to be adopted. A purpose of Section II is to describe various techniques for redundancy as well as to illustrate where each technique is most applicable.

Section III collects and presents the results of a set of brief explorations of the properties of recursive triangular networks. It is divided into three distinctly disjoint subsections. The first establishes that in the case of the three-input rectifier "and" gate (which we studied in detail in Special Scientific Report No. 1) triangular recursion can be used to improve the reliability of the gate no matter how unreliable the rectifiers are. The second subsection deals with attempts to make use of the recursive triangle to improve the reliability of a nonsymmetric Boolean function (ABvCD). Both rectifier and threshold realizations are investigated. This investigation indicates that the most promising level at which to apply recursive triangulation is the basic "and" gate, "or" gate level; where improvements of reliability can be obtained as discussed in Special Scientific Report No. 1. Finally, subsection three deals with attempts to utilize the technique to improve the reliability of the "nor" function. Improvements in reliability were obtained, similar to those reported in Special Scientific Report No. 1. Maximum improvements are obtained with three "nor" gates feeding an "and" gate. However, when the probability of complete failure of the "nor" gate becomes large, a majority gate or "or" gate will give better improvements.

In Section IV, the feasibility of introducing self-repairing capability to a logical net is discussed. In complex logical systems having millions of component units, as also in Satellite systems, it will be very desirable to have a logical organization capable of 'self-repair'. Also, one would like to have a net take preventive measures to avoid any failure that may be anticipated.

A logical scheme for organizing logical nets having the above capabilities has been briefly studied. In this scheme a logical net will be provided with the facility to repeat a calculation with an alternate net (subnet) whenever an error is detected. Substitution of a net (subnet) may also take place at the anticipation of an error occurrence. In such a scheme, greater reliability of operation of a net would be achieved at the cost of increased computational delay. As recalculation will occur

only at the incidence of an error, the net will be making optimum use of the computational time. Also, as the network deteriorates due to, say, aging, its 'reliability' will not decrease, — instead it will take more time to do a computation.

If it is required that each computation done by the net should take only the shortest necessary delay, then one finds that this substitution scheme offers the only possible way of introducing self-repairing capability to the logical nets. In the context of this scheme the problems that arise in the organization of such nets are pointed out. Also, some examples of such nets are discussed to illustrate the techniques of analysis.

II. REDUNDANCY TECHNIQUES FOR SWITCHING NETWORKS

by S. Y. Levy

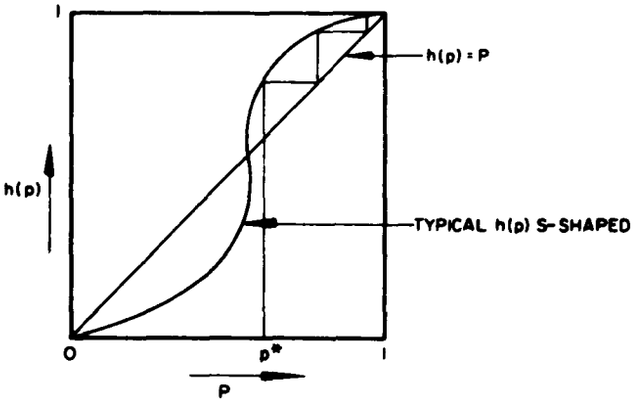
This report is an outgrowth of the Symposium on Redundancy Techniques for Computing Systems which took place in Washington last year. It is intended to survey the field with the dual purpose of seeing what is being done outside of our work at RCA Laboratories to apply redundancy to increase reliability and evaluating our recursive triangle scheme in the light of what we have observed. We discuss those methods which are most relevant to the problems that we ourselves are considering. (In particular, we discuss the papers presented at the Symposium).

In the case of devices built of few components or at least easily accessible components, it has almost always been the policy of manufacturers to increase reliability by either derating components or using more reliable ones. By and large, improvements in reliability have come about through improvement in the reliability of the components used, as shown by most people's experience with their home radios, for example. But increased reliability of components has led to attempts to build still larger more complex systems which need the increased component reliability merely to maintain the old levels of system reliability (you might recognize this as a sort of "Parkinson's law" of technology which says that advances in technology result in an expansion of applications which require the new technology). In addition, many applications of electronics are such that the cost of error (or failure) is high enough to warrant large additional expenditure to prevent it. Applications have arisen for equipment in hostile environment (e.g., radiation belts, war, etc.) where it is desirable to maintain operation in spite of the certainty that parts of the device will fail — or the device will be expected to operate in an environment where repair or replacement of parts is impossible (e.g., an unmanned satellite). At the other extreme there is the new batch-fabricated technology where large quantities of relatively inexpensive devices are made simultaneously and interconnected in a functioning array but with component yields far less than that acceptable from individually produced devices. Can redundancy be applied to this technology to make use of these less-than-perfect arrays of components? At any rate, it is abundantly clear that there is need for suitable techniques for the efficient application of redundancy.

The Reliability of Coherent Systems by Esary and Proschan (Boeing Aircraft)[1] is a general discussion of the analysis of the behavior of contact type networks (e.g., relay networks, unipolar transistor networks). An n-contact network is described by an n-component vector -- each component being an indicator of the state of the corresponding network component ($x_i = 1$ indicates the i^{th} component is operating properly, $x_i = 0$ indicates the component is not operating properly). An overall structure function $\Phi(X)$ is defined (where $\vec{X} = x_1, x_2, \ldots, x_n$) such that $\Phi(\vec{X}) = 1$ if the overall network is functioning properly, $\Phi(\vec{X}) = 0$ if the network has failed. The property coherence is defined by three conditions

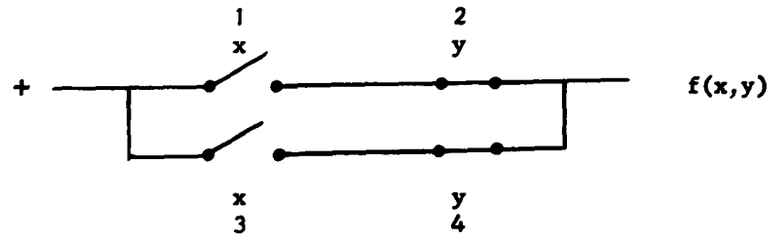
1) $\Phi(\vec{1}) = 1$ where $\vec{1} = 1, 1, \ldots, 1$
2) $\Phi(\vec{0}) = 0$ where $\vec{0} = 0, 0, \ldots, 0$
3) For all $i(x_i \geq y_i) \rightarrow \Phi(\vec{X}) \geq \Phi(\vec{y})$

The paper deals only with coherent nets and their behavior under a form of iteration which involves replacing each contact by a replica of the entire network. To investigate the behavior under this type of recursion they define h(p) as the probability of the network operating properly as a function of p, the reliability of an individual component. h(p) is a so-called "S-shaped" curve.



If each component is associated with a p such that $h(p) > p$, then the iteration results in a staircase effect leading to arbitrarily high reliability (for example, in the diagram $h(p^*) > p^*$; $h(h(p^*)) > h(p^*)$ etc.) If $h(p) < p$ the staircase leads to 0 reliability in much the same way. However, the authors fail to give a reasonable characterization of coherence in terms of what sort of restriction it represents; in fact, they do not give a

single example of a noncoherent network. Winder has pointed out a simple example of a noncoherent structure which we include here.



$f(x,y) = x\bar{y}$

Suppose relay 4 is not functioning properly and is stuck closed while relays 1, 2, 3, are operating correctly.

Then

$$f(x,y) = x\bar{y} \vee x \equiv x \ ,$$

so

$$\Phi(1,1,1,0) = 0$$

Suppose relay 4 is stuck closed and relay 3 is stuck open, while relays 1 and 2 are operating correctly.

Then

$$f(x,y) = x\bar{y} \ ,$$

so

$$\Phi(1,1,0,0) = 1$$

$\Phi(1,1,0,0) > \Phi(1,1,1,0)$ and the network is not coherent.

Note that the structure function $\Phi(\vec{X})$ is a function from binary inputs to a binary output and so may be regarded as a switching function. We can suggest the following characterization of a coherent network.

Theorem: A network is coherent if and only if the switching function $\Phi(\vec{X})$ (not to be confused with $f(\vec{Y})$ the function computed by the network) is positive in all x_i and not identically 0 or 1.

Proof: 1) Assume the network is coherent. Then consider any $\vec{X}$ for which $\Phi(\vec{X}) = 1$. Changing any x_i from 0 to 1 can never decrease $\Phi(\vec{X})$

because the network is coherent. Therefore $\Phi_{\overline{x}_i} - \Phi_{x_i}$ * and the function $\Phi(\vec{X})$ is positive in all x_i.

2) Assume $\Phi(\vec{X})$ is positive, and not identically 0 or 1. Then for all $x_i, \Phi_{\overline{x}_i} - \Phi_x$, this means increasing x_i can never decrease $\Phi(\vec{X})$. Since for at least one value of $\vec{X}$, $\Phi(\vec{X}) = 1$, then $\Phi(\vec{1}) = 1$; and since for at least one value of $\vec{X}$, $\Phi(\vec{X}) = 0$, $\Phi(\vec{0}) = 0$, the network is coherent.

Corollary: Irredundant circuits are coherent. The structure function, Φ, of an irredundant circuit is the "and" function $\Phi(\vec{X}) = x_1 \cdot x_2 \cdots x_n$.

If one considers the mode of failure as failure to operate (to remain stuck in rest position) then all frontal circuits (where relays are normally open) are coherent circuits because a failure can only open a path and make it less likely that the network functions properly.

Though the authors' interest in reliability leads them to attempt an iterative procedure where each element of the network is replaced by a replica of the entire network, the type of circuit which they are studying yields to an entirely different analysis than does the recursive triangle. In the Moore-Shannon[2] paper on redundant contact networks (which incidently devotes a great deal of time to the study of these S-shaped curves) there is an interesting contrast of the two technologies, the mechanical (relay or contact type) and the electronic (rectifier or transistor type). In comparing the Von Neumann[3] redundancy scheme to their own, Moore-Shannon[2] note that where in an electronic system the logical combining ("and", "or", etc.) is subject to error, in the contact system, this is obtained by merely making appropriate connections. On the other hand, in a contact network, the introduction of copies of an input is subject to error (by requiring additional contacts) and a variable is copied in an electronic network by merely making appropriate connections.

---

* $\Phi_{x_i} = \Phi(\vec{X})$ with $x_i = 1$

$\Phi_{\overline{x}_i} = \Phi(\vec{X})$ with $x_i = 0$

Tolerable Errors of Neurons for Infallible Nets by Blum, Ernesto and Verbeek (MIT)[4] is an interesting paper — it would have been better if they included an algorithm for the design of such error-free networks (if indeed the procedure is algorithmic) or at least a little more detail which would substantiate and explain the claims made in the paper. The networks formed consist of n + 1 neurons for an n-input function arranged as shown below:

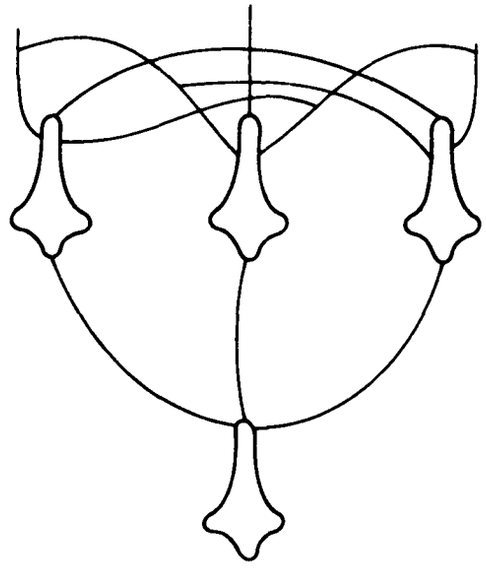


Fig. 2 Three-input neural net

It is somewhat similar to our triangular network except that these nets are nonhomogeneous. No hint is given at a physical model for the type of network they discuss but the error-correcting properties are quite astounding. The nets themselves have been presented before and as such represent nothing new. However, the authors make the interesting comment also made by several others at the conference that the choice of a simple majority element for the combining or output gate is a naive one to be made only in the face of no knowledge about the reliability behavior of the components to be combined (with the exception of a few special cases), and that knowledge about

such factors as the nature of failures, and the bias (if any) of errors can lead to a better choice.

"Codes and Coding Circuitry for Automatic Error Correction Within Digital Systems" by W. H. Kautz (SRI)[5] is one of the better papers presented at the Symposium. Before discussing it, we would like to inject some general comments about the idea of coding for error correction in digital computers. There are two distinct ideas represented here. One is the coding of information to ensure accurate transmission or storage much the same as in communication theory (and the problems are, in fact, the same). The other is the encoding of information to ensure accuracy of computation or logical operation. The former involves encoding of data in the hope that the coded form will resist alteration of the information in the handling, but in the latter the handling is expected to alter the information and the problems are quite different. It is almost exclusively the former with which Kautz deals. The latter problem is discussed in Gore's[6] and Winograd-Cowan's[7] papers presented at the Symposium as well as in many papers not given at the Symposium and will be analyzed in some detail later.

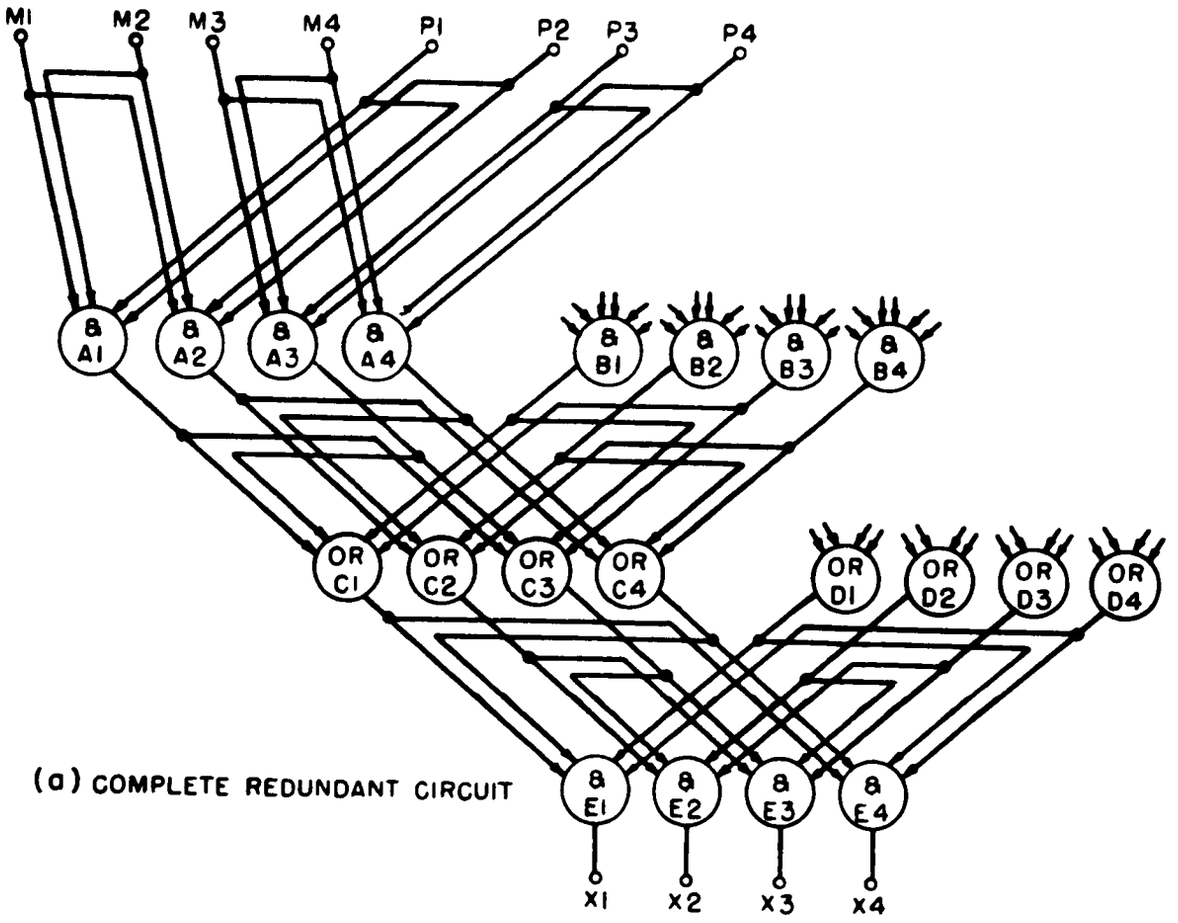
The coding for reliable transmission and storage depends on certain simple properties of the words. Basically, the idea is a simple one -- enough redundant (check) bits are added to each word so that the Hamming distance (i.e., the number of bits in which two words differ) between any two words is $2k + 1$ where k is the largest number of errors which it is desirable to correct for. Then a received word is decoded as the word closest to (least Hamming distance from) the word received. In actual practice a systematic technique for generating code words is used in order to facilitate the design of equipment to perform the task, but the principle remains the same. The coding is generally a form of "block" coding and is encoded and decoded by a set of linear operations (i.e., described in terms of "exclusive or" and its complement). (If the words were to be logically manipulated certain additional constraints would be imposed to ensure that the check resulting from a computation is also a proper check). Kautz is concerned with techniques for physically realizing these codes as well as the quantity of equipment which is actually needed. He points out there are several types of systematic

codes available; the usual group codes, the low density codes which require greater redundancy of information (more bits) but are easier to encode and decode, Berger codes which detect completely biased errors (all 0's or all 1's) and arithmetic codes suitable for checking the arithmetic operations. These latter codes generally derive from number theoretic properties rather than from the logical properties which produce the other codes. [Peterson[8] in a paper on checking an adder demonstrates that the only type of checking possible where the adder and the checking circuitry are independent is some form of remainder mod p check*].

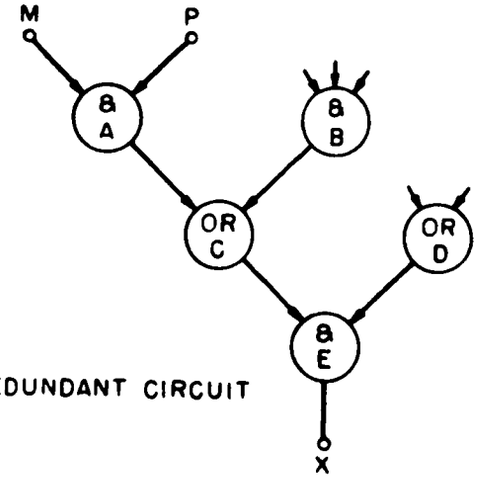
Although it is possible to build multiple error-correcting circuitry, the expense involved is usually so great that it is not done (with the exception of special cases like completely biased errors or bursts of errors in a set of adjacent locations). Kautz points out that "signal redundancy suffers from certain basic limitations: For all but a definable minority of logical circuits, there exist some types of faults whose errors no amount of signal redundancy can correct" (see discussion of noisy computation below). He then goes on to say, "this theoretical limitation need not overly concern us, however, for three reasons. First, this minority includes several circuit operations of considerable practical importance, such as simple data transfer, parity checking, and linear (pure "exclusive or") logical circuits generally. Second we rarely need to protect a circuit against all possible faults, but only a selected class deemed to be most likely. Third, recently it has been shown that, under certain reasonable assumptions, a network can be made arbitrarily reliable with the proper <u>combination</u> of signal and circuit redundancies." [Unfortunately, we have not seen the proof of this third statement].

<u>Quadded Logic</u> by J. G. Tryon (Bell Labs.)[9] presents a technique for introducing redundancy into a logical chain. Essentially, it involves quadruplicating the number of gates and then systematically interconnecting them so as to mask faults. It is Tryon's hope to mask these faults as close to their source as is possible. His approach is a qualitative one — he is not immediately concerned with the question "how much does quadding improve reliability?" —

* A remainder mod p check is performed by carrying as check bits the remainder when the word is divided by p. It then follows that the sum of the checks of two numbers is the check of the sum (similarly the difference, product, quotient in a fashion).



(a) COMPLETE REDUNDANT CIRCUIT



(b) ORIGINAL UNREDUNDANT CIRCUIT

Fig. 3 BASIC QUADDED CIRCUIT

he asks instead - "If the output of this gate is stuck high, how can I correct for it?" If there are two stages of logic beyond the faulty gate, then quadding is an answer; if not, quadding is not an answer.

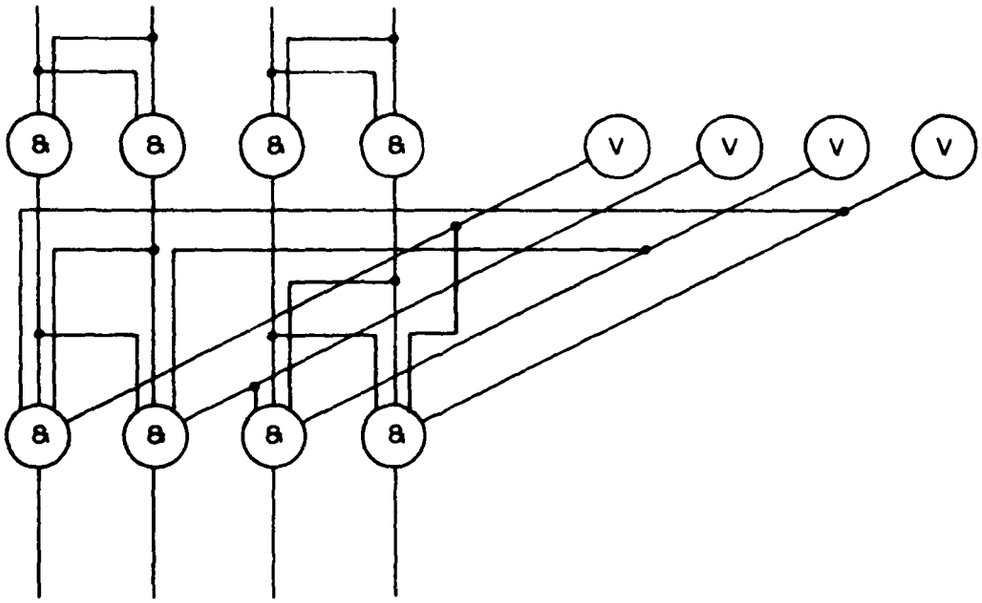
Quadding is applied to a chain (gating or timing) rather than to a single logical element. A quadded chain has four times the number of gates of a nonredundant chain, and each gate in the quadded chain has two times the fan-in and two times the fan-out of a nonredundant gate. The termination of a quadded chain is four lines, and the decision with regard to how to interpret these lines in the case that they are not all in agreement is outside of the domain of quadding.

In his report Tryon presents an algorithm for quadding which we will partially reproduce. Tryon describes the technique for the quadding of combinational logic built of "and" gates, "or" gates, and "inverters" by three rules:

Rule 1. In circuits in which levels of "and" and "or" alternate, wiring patterns must be chosen so that no signal encounters the same pattern twice in succession as it goes along.

So, for example, in the diagram of the quadded circuit the connection pattern between the A and C gates is different from that between the C and E gates.

Rule 2. Whenever an "and" unit feeds an "and" unit (or an "or" unit), all wiring patterns feeding the first unit must be the same as that between the first and second units. An example follows on the next page.

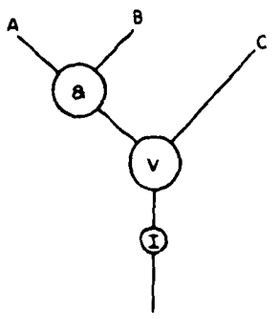


The final rule includes the first two and is:

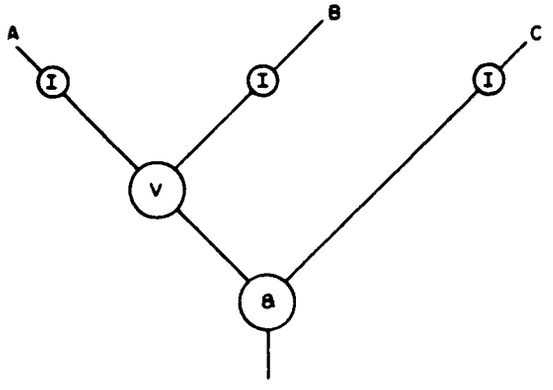
Rule 3. As any signal flows from the input of any "And" or "Or" unit, along any path, the parity of the number of changes in pairing that it encounters must be equal to the parity of the sum of (a) the number of negations and (b) the number of "and"/"or" transitions.

Incidentally, negation is handled by applying De Morgan's laws to move the negations to the inputs, quadding, and then moving the negations back (making the appropriate changes.)

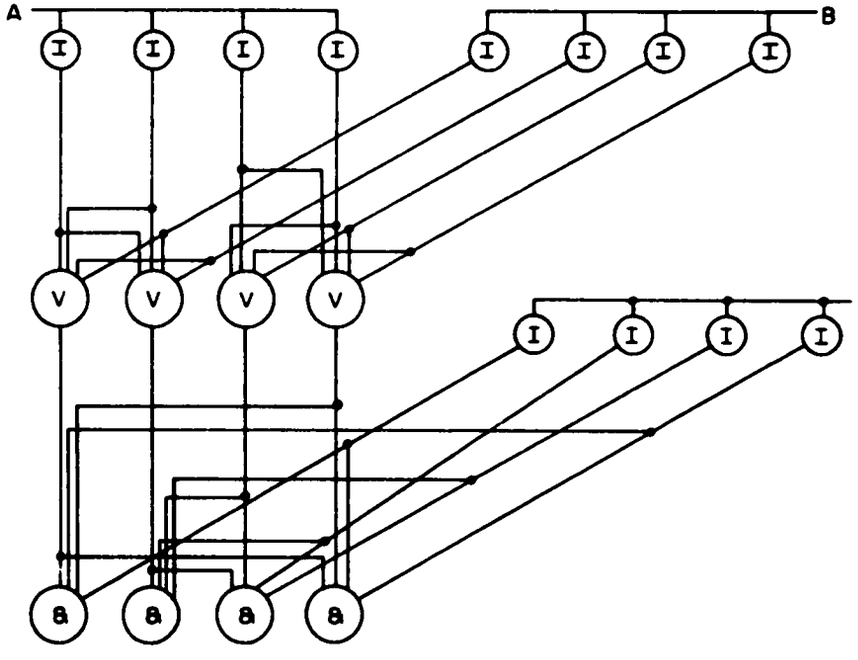
Thus to quad:



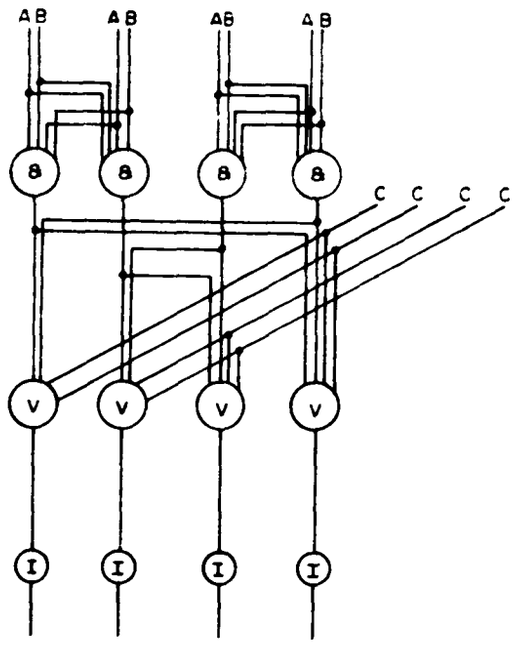
we first change it to:



Then quad it



and return the negations.



We also present an example of a quadded circuit which should give the flavor of the technique. Figure 3 is an example of a circuit in both non-redundant and quadded forms. Consider the quadded version. If as shown all the M inputs are 1's and all the P inputs are 0's, then the output of gates A_1 - A_4 should properly be 0's. Suppose A_4 is malfunctioning in such a way that its output is a 1 instead. Then at the next level of logic ("or" gates C_1 - C_4) this incorrect 1 spreads to 2 gates C_1 and C_3 (assuming B_1 - B_4 have output 0). But at the following level of "and" gates, the situation is corrected and the output is the proper "0" on all 4 lines. It is fortunate that the "or" gates C_1 - C_4 were not the final level in the chain. Suppose, on the other hand, that all the M's and P's are 1's and that the outputs of the gates A_1 - A_4 are proper 1's, but that the output of C_4 is an incorrect 0 (while C_1 - C_3 put out 1's); then the output of E_1 and E_4 will be 0 while the output of E_2 and E_3 will be 1. The required action is unclear!

This demonstrates that quadding requires at least two levels beyond the source of a single error in order to correct for it; — this implies that quadding is more effective as the chain increases. This would, in most

technologies, prove a serious drawback since a) a chain terminates at a storage element (flip-flop, memory, etc.), and b) the usual tendency in design of a system is to try to keep the chains as short as possible to avoid deterioration of signal and increased propagation time.

Tryon has managed to avoid these problems since he quads memory, timing devices and even flip-flops. (The latter through an interconnected "nor" gate type construction of the flip-flops)

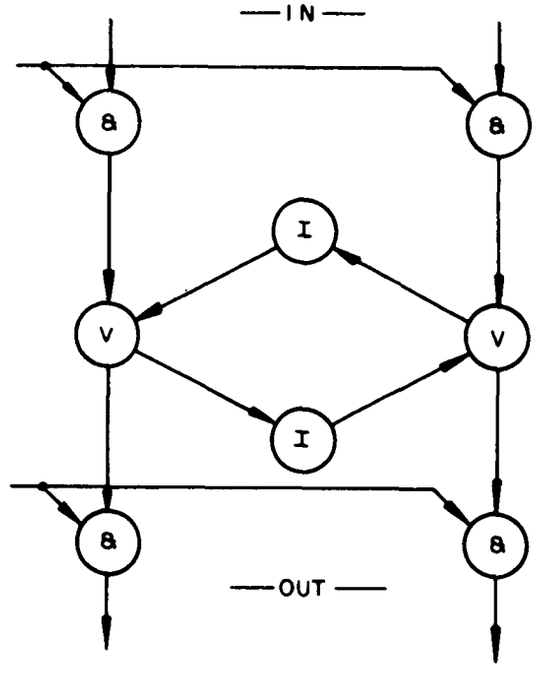


Fig. 4 Tryon flip-flop. It is composed of combinational elements that may be quadded in the usual manner.

The fact is that quadding is distinctly oriented to a particular type of equipment. (There is nothing wrong with this; in fact, we believe that the type of redundancy to be used should be designed to fit the particular technology. We state it only because we believe that it should be noted).

Tryon goes further in the design of his technique than anybody else whose work we have read. He has given serious thought to the problem of maintenance of a redundant system. In a system which is designed to mask faults, it is harder to detect faults. Consequently, it is more difficult to locate failures, and so a redundant machine would be more likely to accumulate them than would a nonredundant machine in which they are almost immediately detected. As a result there is a crossover point in time where if both a nonredundant and a redundant machine are still operating, the redundant machine is more likely to fail. B. R. Saltzberg, a co-worker of Tryon's, has devised a technique for isolating nonredundant segments of a quadded network in order to check for failed parts. This technique (as an interesting extra) demonstrates how redundant power supplies may be used to protect the system against supply failures as well as logical failures.

Quadding as a technique is applicable to relatively good components. In the face of reasonably high probability of multiple failure in a quad, it is not effective. For the same reasons it is better suited to a situation where regular maintenance will not allow multiple faults to accumulate. However, it does offer a method of introducing redundancy at a relatively reasonable price (about eight times that of a nonredundant machine) and in a substantially uniform way throughout the machine.

Adaptive Vote Takers Improve the Use of Redundancy by W. H. Pierce (Westinghouse)[10] presents a variation on the majority logic scheme of Von Neumann. In this technique certain of the replicated organs are assumed to be more likely to be incorrect than are others, and so less credibility is attached to their decisions. In effect, a more consistently reliable record is rewarded by a greater say in the final decision-making element, and as failures occur the voting weights are continually adjusted.

This is the most complex system for introducing redundancy which we have seen. Since one of the great appeals of a binary system is its relative immunity to error, we find the introduction of the analog weight adjustment a possible weakness. In addition we do not see the need for adjusting the weights over a continuum since the threshold decision-making device will recognize only a relatively small number of different functions of the inputs — perhaps adjusting the weights discretely would help the reliability of this involved

device. Also, the Pierce scheme is not intended to improve the computation part of the system, but is directed only at the decision maker. It introduces an additional large (unchecked) system which appears to be another likely source of error. Although we find the idea interesting and the analysis a really intricate piece of work, we also find it impossible to consider it seriously as a technique for increasing reliability.

W. H. Mann (Westinghouse) in Restorative Processes for Redundant Computing Systems[11] presents some very good solid thoughts on redundancy. The first section of the paper discusses modes of placing restoring organs into a system at desired intervals. He discusses the various effects of the distribution of these organs on the system reliability. His discussion is really really outstanding when he discusses common assumptions about components and the way they fail and how these assumptions affect reliability analysis. The most interesting of these comments follows:

Assumptions and their effects

1). The effects of highly improbable failure modes are negligible.

The flaw is that a failure which is improbable at the circuit level may be highly probable at the large system level. For example:

Basic nonredundant system

System size	1000 stages
Signal processor failure probability	0.0005
Nonredundant system failure probability	0.394

All failures capable of restoration - Redundant system

Restoring circuit failure probability	0.001
Redundant stage failure probability	3.36×10^{-8}
System failure probability	3.36×10^{-5}

Redundant system with additional 0.1 percent of failures not capable of restoration

Signal processor failure probability	0.0005005
Restoring checked failure probability	0.001001
System failure probability	750.5×10^{-5}

2). A false 1 is as likely as a false 0. This is the type of assumption which would lead to a simple majority decision-making element. If, for example, one considers a model like ours of the rectifier "and" gate where the open diode acts as a 1, then the outputs of "and" gates are such that all output 0's are correct but some output 1's are incorrect. If this is the case an "and" gate is the best choice for a combining element. If the probability of a false 0 becomes positive the choice of combining function should move away from "and" on the lattice of Boolean functions going close to majority as the probability of 0 and 1 get closer to equal and finally towards "or" as the probability of 0 gets larger than that of 1.

Information Theory and Redundancy

This section will cover work reported over a long period of time by many people. In pre-information theory days, the standard procedure used to transmit binary information through a noisy channel was to repeat the transmitted signal 2n+1 times. Then at the receiving end a majority vote would be taken to determine the nature of the transmitted signal. If n + 1 or more 1's were received the signal would be interpreted as a 1, otherwise the signal would be interpreted as a 0. This is a simple example of an n-error-correcting code (since if there are n or fewer errors the signal would still be interpreted correctly). This obviously gives an increase in reliability, but at a price — a decrease in the rate of transmission of information. [To be n-error-correcting causes a decrease in transmission rate to $\frac{1}{2n+1}$ times that of the noncorrected case]. The probability of error for this particular system is given by $p_e \sim 2^{-k/R}$ where R is the rate of transmission of information. By decreasing the rate (R), p_e can be made to decrease, but to make p_e arbitrarily small it is necessary to make R arbitrarily small as well. Information theory provided a different solution to the problem of increasing the reliability of transmission of information. If one defines a quantity C (the channel capacity) as a maximum rate at which it is possible to obtain mutual information from the output of the channel about its input, then by a type of coding called block coding it becomes possible to transmit information through that channel with a probability of error given by $p_e = \begin{cases} 1 & R < c \\ 0 & R > c \end{cases}$ where R is again the rate of transmission. [From this relation we see that C is also the maximum rate at which it is possible to receive information over the channel with arbitrarily high reliability]. To block code, take the sequence of bits to be transmitted, break it into units k bits long, and encode each k bit sequence into

an n bit sequence ($n > k$); then transmit the n bit sequence. At the receiving end, the n bit sequence distorted by noise is decoded into the proper k bit sequence. If R, which equals k/n, is kept constant and less than C, then by increasing both k and n the error probability may be made arbitrarily small.

But all the results we have presented thus far hold only for the case that the information coming out of the decoder is intended to be the same information which went into the encoder. But what of the case where the information is not just being transmitted or stored, but is being logically combined with other information (also appropriately coded); can block coding provide a technique for obtaining arbitrary reliability in some more efficient manner than the iterative methods mentioned above? Von Neumann, in his paper on probabilistic logics,[3] expressed the opinion that it could, but just how eluded him. Elias[12] noted that the techniques of replication and majority vote, or iteration to improve reliability are much like the pre-information theory techniques for reliable signal transmission. If, instead of building a redundant computer operating at a higher reliability level, one chooses to use the redundant elements to build extra computers operating at the old reliability levels, then one could perform more (though less reliable) computation. So in a sense the result is a trade-off of computation rate for reliability. Elias set about investigating this question, by considering information to be combined according to any of the 16 binary functions of 2 binary inputs. In particular, he studied the "and" function (and then discussed its generalization). Since many of the later papers refer specifically to his results, we shall reproduce his ingenious proof here.

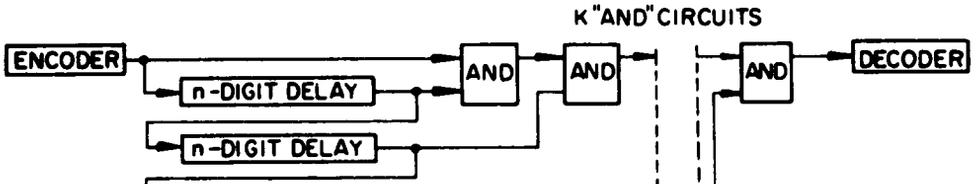


Fig. 5 Multiple "and" computer

The "and" circuit shown above takes the "and" of k successive n digit blocks of coded input, using k-1 n digit delays and k-1 noisy "and" gates. We call the input sequence consisting of k ones 1^k, and the sequence consisting of k-j ones and j zeros $1^{k-j}0^j$ etc. The n bit block which is the coded version of s will be denoted by T(s), so $T(1^k)$ is the sequence which comes out of the encoder when a sequence of k ones is put in. Consider the following set of n-digit blocks which are possible results of this computation as they come out of the last "and" circuit before decoding:

$$T(1^k) \cdot T(1^k) \ldots\ldots T(1^k) \cdot T(1^k)$$
$$T(1^k) \cdot T(1^k) \ldots\ldots T(1^k) \cdot T(1^{k-1}0^1)$$
$$T(1^k) \cdot T(1^k) \ldots\ldots T(1^{k-1}0^1) \cdot T(1^{k-2}0^2)$$
$$\vdots$$
$$T(1^k) \cdot T(1^{k-1}0^k) \ldots T(1^10^{k-1}) \cdot T(0^k)$$

Each line can be obtained from the preceding line by "anding" with one new factor, since the "and" of $T(1^k)$ with itself any number of times is still $T(1^k)$. But adding a new factor by "anding" can only strike out some of the ones which are present in a sequence. All of the n digit sequences above must be decoded differently, since they all represent computations having different results (the proper result for each row is the argument of the rightmost factor). In going from one line to the next we eliminate at least one one. If we let d_i be the number of ones eliminated when passing from line i to line i+1, then, since there are at most n ones in the first line and at least no ones in the last line, we have

$$\sum_{i=1}^{k} d_i < n; \qquad d_i > 1$$

It follows that if $n < 2k$ some of the d_i will be just one and a single error in the output can cause two adjacent lines to become confused, so that the decoder will print out the wrong answer when such an error is made, if it prints out the right answer when no errors are made.

From this it is concluded that:

1) It is not possible to get a reliability for the block greater than that present in the individual devices (reliability here being probability of correct output) until the rate of transmission drops from unity($k = n$) to $\frac{1}{2}$ ($2k = n$) and even then there are single errors which the decoder can detect but only attempt to correct, with probability $\frac{1}{2}$ of guessing wrong.

2) The rate must drop to $\frac{1}{3}$ ($3k = n$) before all single errors in the computations shown above can be corrected. Notice that $n = 3k$ has a familiar ring and, in fact, if we must go to that much redundancy we could as well and certainly as easily (from the point of view of decoding and encoding) go to simple iteration of the i^{th} digit of each block. So at least as far as minimum distance between words between codewords is concerned, no block code (for the computer considered) does better than simple iteration of the input digits [and as mentioned earlier this leads to a capacity which is zero].

Elias considered all the possible binary functions of 2 inputs x_1 and x_2. Of these, he conjectured that the only nondegenerate functions to which coding could be applied to achieve arbitrary reliability were "exclusive or" and "equivalence". But "exclusive or" and "equivalence" and any combination of the degenerate functions of 2 variables do not form a complete set of propositional functions. There are Boolean functions like "and", "or" which can not be constructed from these functions, and so, in all likelihood, one could not construct a complete general purpose computer from these block-coded functions.

Peterson and Rabin[13] prove the same results as Elias under somewhat more relaxed restrictions, and in fact they are able to verify (algebraically) Elias' conjecture. Their results are summarized here:

1) For single-error detection codes which consist of the original information with a check symbol, there is no simpler system than making the check symbols duplicates of the information for any nontrivial logical operations except "exclusive or" and "equivalence" both of which may be checked with "parity digits".

2) For general block coding with the output decoding one — one in the absence of errors, and the coded blocks processed digit by digit, for "and", "or", "exclusive-or", "equivalence" the same code must be used at the output as at the inputs (assuming that in each case the sequence of all zeros codes into the sequence of all zeros). For the other six nontrivial logical operations the input and output codes are closely related. For all logical operations except "exclusive or" and "equivalence", the operation done on the coded blocks must be the same as the operation being checked. For "exclusive or" and "equivalence", each digit in the coded blocks is a parity check on some subset of the digits of the uncoded block. In other words, group codes and only group codes can be used to check these two operations.

3) For the same restrictions on coding as on (2), and for all nontrivial logical operations except "exclusive or" and "equivalence", there is no simpler coding system with a specified ability to detect or correct errors than a system in which the coded sequence consists of a number of copies of the uncoded sequence.

Relying heavily on the previous results of Elias and Peterson-Rabin, Winograd[14] extended these results to the m variable case ($m \geq 2$). His results are summarized "Of all the 2^{2^m} Boolean functions of m variables, only 2^{m+1} functions are linear [and hence may be block-coded to advantage]; namely, all functions which can be represented as $f(x_1, \ldots, x_m) = k_0 + \sum_{i=1}^{m} k_i x_i$ for some $k_i = 1$ or 0. Of those 2^{m+1} functions only 2 are explicit functions of all m variables, namely

$$\sum_{i=1}^{n} x_i \quad \text{and} \quad 1 + \sum_{i=1}^{n} x_i \text{ ."}$$

where Σ and + stand for "exclusive or"

Elias also pointed out that if we relax certain restrictions such as insisting that the decoding at the output be one — one in the absence of noise then we can do some of the computation in the error-free encoder and decoder and improve our reliability. To this technique Elias applied the rather graphic name, "cheating".

Again it may be said that this does not mean that only iteration serves to check computer operations. If extra logical information is available about the operation being performed, then an appropriate check may often be designed, as, for example, a remainder modulo p check on an arithmetic unit. In addition, if one thinks in terms of applying the checks to code the function as well as the inputs and in addition "owns up" to the possibility of error in the encoder and decoder then the coding may result in improved reliability (though not arbitrarily high reliability).

Recursive Triangular Nets[15,16]

The recursive triangular network has been described in previous reports. It is, like the Shannon-Moore[2] procedure an iterative technique for applying redundancy and consequently the quantity of equipment used grows exponentially with the number of stages of recursion. Very generally, the technique of triangular recursion is this:

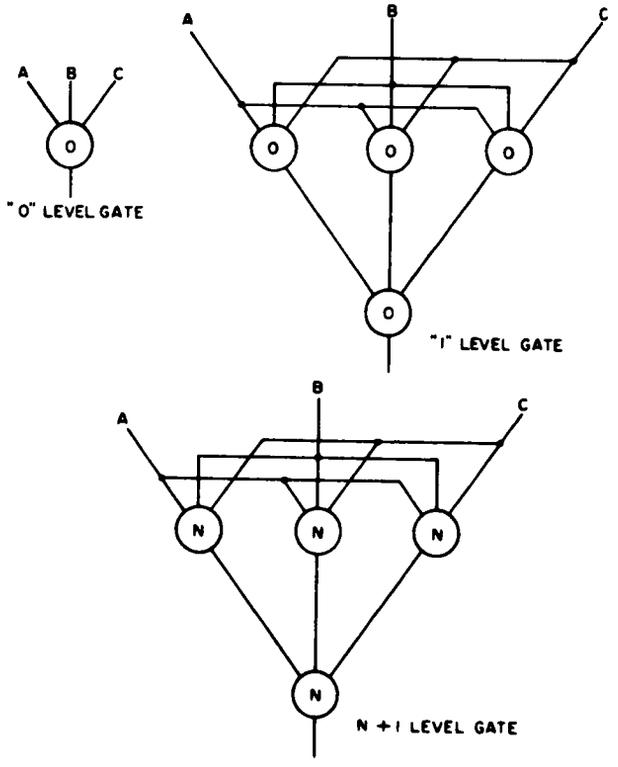


Fig. 6 Triangular recursion method

Where the basic gate (whose reliability is to be improved) is the 0 level gate the 1 level gate is formed by a triangle of n (the number of inputs — in Fig. 6 n = 3) 0 level gates feeding a 0 level gate; and in general, an N + 1 level gate is formed by feeding n N^{th} level gates into an N^{th} level gate. The manner in which a gate fails as well as the way in which it operates are factors which determine whether or not triangular recursion can be used to improve the reliability of a gate. Our analysis of the reliability of a gate derives from the idea that a failed gate computes a logical function other than the one for which it is designed. This other logical function may be a degenerate one insensitive to some of the input variables (or even all the input variables as in the case of a gate stuck at 1 or 0), or it may just be a different function of all of the variables. A set of recurrence relations relate the probabilities of occurrence of each of these failure functions at a stage of recursion to the value of these probabilities at the previous stage. In earlier reports these relations have been produced and analyzed for various probability distributions over failure and design functions. Under certain restrictions on the distributions, it is possible to continue recurring until any arbitrary degree of reliability is achieved, but with most realistic distributions, a maximum value of reliability is reached after just a few stages of recursion, and if recursion is continued beyond the maximum, reliability will drop off, eventually falling to zero. In the case of the simple rectifier n-input "and" or "or" gate the probability of correct functional operation goes from $(1-p)^n$ (where p is the probability of failure of a rectifier) in the basic rectifier gate to very close to $1-p^n$ in a few stages of recursion. This would mean, for example, that for a 4-input gate with a probability of rectifier failure of .01 that the probability of failure of a gate decreases from 3.94×10^{-2} in the nonredundant gate to 10^{-8} in the redundant gate or by a factor of about 25000. Of course, the price paid for this would be very high indeed — it would require 625 times the equipment in a nonredundant gate. If an improvement factor of about 20000 would suffice, this could be obtained at a cost of about 125 times the nonredundant equipment. These quantities of equipment sound forbidding. They are (although Von Neumann spoke of numbers on the order of 1000 to 25000 in his multiplexing technique). Even if the economic considerations were not by themselves sufficient to bar such a system, certainly considerations of power consumption and bulk would exert a strong negative influence.

However, considerable improvement in reliability is obtained at the early stages of recursion where the amount of equipment required is not nearly so large as in the examples cited above. The table below gives some examples of improvement at the early stages of recursion.

	Number of Inputs	p	N Stage of Recursion	P_N P(failure) at N^{th} Recursion	$\frac{P_0}{P_N}$	Equipment ratio $\frac{\text{redundant}}{\text{nonredundant}}$
1)	3	0.2	0	.488	1	1
			1	.112	4.4	4
			2	.011	44	16
			3	.009	54	64
2)	3	0.01	0	3×10^{-2}	1	1
			1	3×10^{-4}	100	4
			2	3×10^{-9}	10,000,000	16
3)	4	0.3	0	.76	1	1
			1	.2	3.8	5
			2	.011	7	25
			3	.009	8.4	125

Notice that case 2, which is the most striking of the 3 cases, represents a gate made from components which are clearly worse than anything commercially acceptable today — rectifiers which fail 1 out of every hundred times they are called on — (and cases 1 and 3 are considerably worse than that). The point made here is that the triangular recursive redundancy is applicable to really poor components — techniques like, for example, quadding are certainly more reasonable to use in the case that the likelihood of multiple error is small indeed, but in the case where the components are really bad and the occurrence of multiple error is not unlikely, then only the recursive triangle of all the techniques discussed is applicable.

In summary, the recursive triangle requires large amounts of redundancy, but it is able to make extremely efficient use of the "extra" components. If constructed with components with a low error rate recursive triangular nets might be used to build equipment with an extremely long lifetime (e.g., for use in a space craft taking a long journey). On the other hand the recursive triangle might be considered as applicable to a situation where a great number of components can be produced cheaply (and perhaps not too reliably), but it is still desired to construct a useful system from them.

The present

The current 'state of practice' of redundancy techniques as applied to computing systems is yet far from anything discussed in this report. So far as I can gather, people design conservatively and stock spare parts (usually in the form of 'plug-in packages') on the one hand, and where the application (and financial capability) warrant it, as, for example, in satellites or defense systems like SAGE , they introduce entire standby systems to be operated in parallel or as they are required. Redundancy applied at levels lower than duplication of the entire system is usually present in the form of extra parity bits for detection of errors in information transfer and remainder-modulo-some-p checks for detection of error in arithmetic computation, with correction of the errors provided by repeating any operation shown to be in error.

Increased computation capability is a concept very easy to comprehend, but the concept of increased reliability is not so well understood. In fact, the former is regarded as a goal of the system while the latter represents an obstacle (i.e., the enemy). Consequently the two have been separated, and people find it easy to justify large additional expense for the former but far more difficult to justify it for the latter. Of course, these two are not at all independent, but until many more people come to think in terms of the ways in which they are related, the systems which we have discussed in this report will be slow to be adopted.

The most ambitious that anyone has been getting at present (to the best of our knowledge) is to attempt some of the techniques discussed in Kautz'[5] paper which do not require large additional quantities of equipment. If faced with an appropriate technology (and-or-not-quite reliable) then a technique similar to what Tryon[9] suggests could be the next step. If on the other hand the technology calls for making use of very cheap but unreliable components, then a very good case could be made for adopting something like the recursive triangle. And between these two techniques, cases are to be made for many of the systems which have been discussed.

III. SOME THOUGHTS ON RECURSIVE TRIANGULAR NETWORKS

by S. Y. Levy

This report presents the results of a set of brief investigations of the properties of recursive triangular networks. It is divided into three distinctly disjoint sections. The first establishes that in the case of the three-input rectifier "and" gate (which we studied in detail in Special Scientific Report No. 1) triangular recursion can be used to improve the reliability of the gate no matter how unreliable the rectifiers are. The second section deals with attempts to make use of the recursive triangle to improve the reliability of a nonsymmetric Boolean function (ABvCD). Both rectifier and threshold realizations are investigated. Finally, section three deals with attempts to utilize the technique to improve the reliability of a complete function "nor".

PART I THE 3 INPUT RECTIFIER GATE

In the three input rectifier gate the probability of correct operation $p\{ABC\} = S = (1-p)^3$ where p is the probability of a rectifier malfunctioning. In this section we will show that no matter what value p assumes in the region (0,1), the first stage of recursion is always more reliable than the basic gate ($S_1 > S_o$).

Using the notation introduced in the previous reports $p\{T\} = T$, $p\{A\} = p\{B\} = p\{C\} = Q$, $p\{AB\} = p\{AC\} = p\{BC\} = R$, $p\{ABC\} = S$, we write the recurrence relation $S_1(Q_o, R_o, S_o, T_o)$.

$$\begin{aligned} S_1 &= 3Q_oS_o + 18R_o^3 + 3R_oS_o^2 + 6R_oS_oT_o + 18Q_oR_o^2 \\ &+ 18Q_oR_oS_o + 18R_o^2S_o + 6Q_o^2S_o + 24R_o^3S_o + S_o^4 \\ &+ 18R_o^2S_oT_o + 3S_o^2T_o^2 + 3S_o^3T_o + 63Q_o^2R_oS_o \\ &+ 63Q_oR_o^2S_o + 27Q_o^2S_o^2 + 9Q_oS_o^3 + 27R_o^2S_o^2 \\ &+ 9R_oS_o^3 + 18Q_oR_oS_oT_o + 18Q_oS_o^2T_o \\ &+ 18R_oS_o^2T + 54Q_oR_oS_o^2 \end{aligned}$$

Substitute $Q_o = p^2(1-p)$, $R_o = p(1-p)^2$, $S_o = (1-p)^3$, $T_o = p^3$ into the above equation.

Then since we seek the region where $S_1 > S_o$ we divide S_1 by $S_o = (1-p)^3$. The polynomial which results is:

$$1 + 4p + 6p^2 - 12p^3 - 6p^4 + 21p^5 - 66p^6 + 108p^7 - 72p^8 + 71p^9.$$

It is plotted on the graph where it appears rather clearly that this ratio is always greater than 1, for $(0 < p < 1)$. Since the improvement which results from recursion is greater as the number of inputs increases it is expected that the result holds for all $n \geq 3$ (and in fact Maitra has proved the result for n = 2 as well).

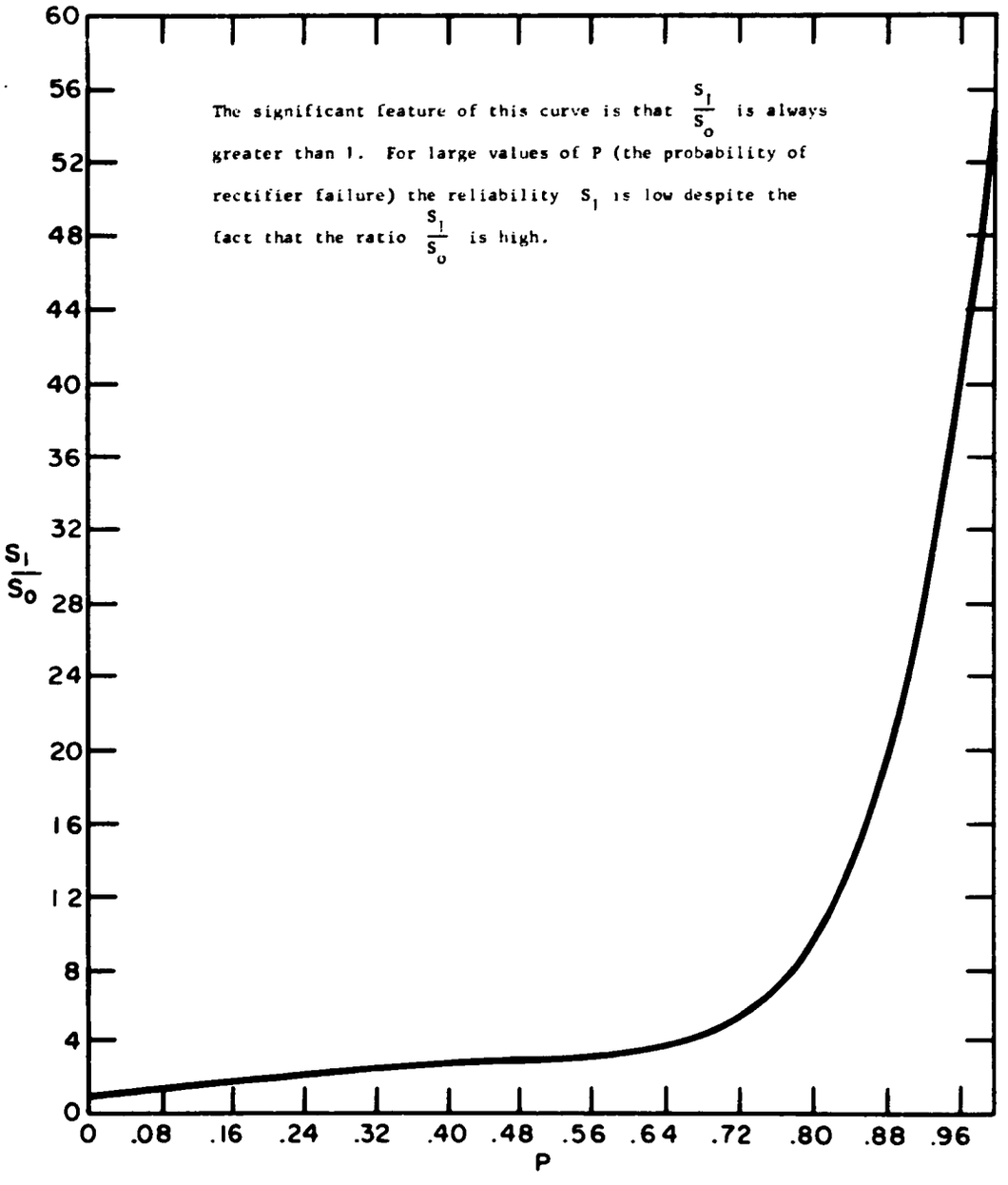
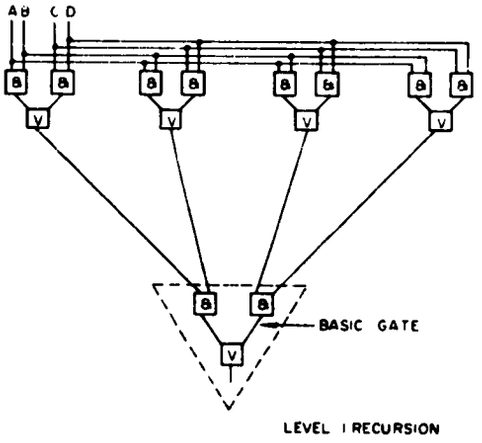


Fig. 7 **Improvement ratio at first level of recursion (3-input rectifier gate)**

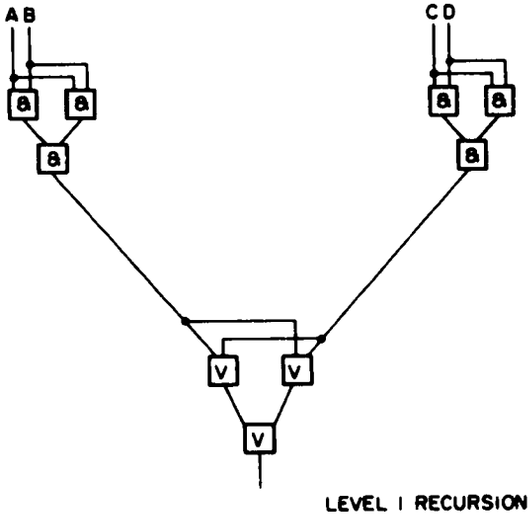
PART II. THE FUNCTION ABvCD

Suppose first that it is desired to synthesize the function ABvCD from a network of rectifier gates. Suppose further it is desired to make use of triangular recursion to improve the reliability of these gates. How should the recursion be applied? There are two obvious alternatives.

1) Apply triangular recursion to a basic triplet consisting of two "and" gates feeding an "or" gate



2) Apply triangular recursion to each of the gates which compose the basic triangle



Then first consider Case 1. Tabulate the behavior of a zero level gate.

Number of failures	p{n failures}	Symmetry type resulting from these failures	Number of members of that symmetry type which can occur as a result of n failures
0	$(1-p)^6$	ABvCD	1
1	$p(1-p)^5$	BvCD	4
		AB	2
2	$p^2(1-p)^4$	T	2
		AvC	4
		A	4
		F	1
		AB	4
3	$p^3(1-p)^3$	T	6
		F	4
		A	8
		AB	2
4	$p^4(1-p)^2$	T	5
		F	6
		A	4
5	$p^5(1-p)$	F	4
		T	2
6	p^6	F	1

The probabilities associated with the different operating states at the basic triangle level may be computed easily with the aid of this table. Since the purpose of this report is only a brief inspection of this problem we will assume a value for p, and compute the probabilities.

$$p = 0.1$$

The distribution of operating states in the basic gate is:

$$p\{ABvCD\} = (1-p)^6 = .53$$

$$p\{AvCD\} = p\{BvCD\} = p\{ABvC\} = p\{ABvD\} = p(1-p)^5 = .059$$

$$p\{AB\} = p\{CD\} = p(1-p)^5 + 2p^2(1-p)^4 + p^3(1-p)^3 = .073$$

$$p\{AvC\} = p\{AvD\} = p\{BvC\} = p\{BvD\} = p^2(1-p)^4 = .007$$

$$p\{A\} = p\{B\} = p\{C\} = p\{D\} = p^2(1-p)^4 + 2p^3(1-p)^3 + p^4(1-p)^2 = .008$$

$$p\{T\} = 2p^2(1-p)^4 + 6p^3(1-p)^3 + 5p^4(1-p)^2 + 2p^5(1-p) = .018$$

$$p\{F\} = p^2(1-p)^4 + 4p^3(1-p)^3 + 6p^4(1-p)^2 + 4p^5(1-p) + p^6 = .010$$

With the aid of the following tables we can compute the recurrence relation which will tell us the probability distribution of the operating states in a first level recursion. Each of the following charts represents the resultant function of the first level triangular gate when the apex gate is a particular function (fixed for each chart) and the inputs of this apex function are specified by the left column and the heading of each column. [These charts are read like the mileage charts on road maps]. For example,if the apex gate is performing the function A' B' (see chart I) and lines A' and B' are the function ABvD (the next to the last row) and ABvC (the third column from the right) respectively, then at their intersection ABvCD is the resulting function. Notice also that charts I and II are half shown. Therefore, when totalling combinations, each entry with the exception of those along the main diagonal is assumed to appear twice.

In order to calculate p{ABvCD}, it is first necessary to compute the probabilities of the states which result from a conjunction of 2 lines feeding the apex gate. These are easily taken from chart I. (Notation: we will use the name of the function to stand for its probability -- thus (AB) is p{AB}).

First, we will calculate probabilities of a network performing a particular function when the apex gate is performing a conjunction.

These probabilities are conditional probabilities - actually (AB) = p{AB | Apex gate is conjunction} and similarly for the other probabilities.

$$(ABvCD) = (ABvCD)[2[4(AvC) + 4(AvCD) + T) + (ABvCD)] + 4(AvCD)(BvCD) = .646$$

$$(AB) = (CD) = 2(AB)[(T) + 2(A) + 4(AvC) + 4(ABvC) + (ABvCD)] + (AB)^2 + 2(A)(B) = .126$$

T																
F	F															
A	F	A														
B	F	AB	B													
C	F	AC	BC	C												
D	F	AD	BD	CD	D											
AB	F	AB	AB	ABC	ABD	AB										
CD	F	ACD	BCD	CD	CD	ABCD	CD									
AvC	F	A	B(AvC)	C	D(AvC)	AB	CD	AvC								
AvD	F	A	B(AvD)	C(AvD)	D	AB	CD	AvCD	AvD							
BvC	F	A(BvC)	B	C	D(BvC)	AB	CD	ABvC	ABvAC vBDvCD	BvC						
BvD	F	A(BvD)	B	C(BvD)	D	AB	CD	ABvADv BCvCD	ABvD	BvCD	BvD					
AvCD	F	A	B(AvCD)	C(AvD)	D(AvC)	AB	CD	AvCD	AvCD	ABvAC vCD	ABvAD vCD	AvCD				
BvCD	F	A(BvCD)	B	C(BvD)	D(BvC)	AB	CD	ABvBC vCD	ABvBD vCD	BvCD	BvCD	ABvCD	BvCD			
ABvC	F	A(BvC)	B(AvC)	C	D(ABvC)	AB	CD	ABvC	ABvAC vCD	ABvC	ABvAC vBDvCD	ABvAC vCD	ABvBC vCD	ABvC		
ABvD	F	A(BvD)	B(AvD)	C(ABvD)	D	AB	CD	ABvAD vCD	ABvD	ABvBD vCD	ABvD	ABvAD vCD	ABvBD vCD	ABvCD	ABvD	
ABvCD	F	A(BvCD)	B(AvCD)	C(ABvD)	D(ABvC)	AB	CD	ABvCD	ABvCD	ABvCD	ABvCD	ABvCD	ABvCD	ABvCD	ABvCD	ABv·

Chart I. Apex Gate performs A' B' (or C' D')

F															
T	T														
A	T	A													
B	T	AvB	B												
C	T	AvC	BvC	C											
D	T	AvD	BvD	CvD	D										
AB	T	A	B	ABvC	ABvD	AB									
CD	T	AvCD	BvCD	C	D	ABvCD	CD								
AvC	T	AvC	AvBvC	AvC	AvCvD	AvC	AvC	AvC							
AvD	T	AvD	AvBvD	AvCvD	AvD	AvD	AvD	AvCvD	AvD						
BvC	T	AvBvC	BvC	BvC	BvCvD	BvC	BvC	AvBvC	AvBvCvD	BvC					
BvD	T	AvBvD	BvD	BvCvD	BvD	BvD	BvD	AvBvCvD	AvBvD	BvCvD	BvD				
AvCD	T	AvCD	AvBvCD	AvC	AvD	AvCD	AvCD	AvC	AvD	AvBvC	AvBvD	AvCD			
BvCD	T	AvBvCD	BvCD	BvC	BvD	BvCD	BvCD	AvBvC	AvBvD	BvC	BvD	AvBvCD	BvCD		
ABvC	T	AvC	BvC	ABvC	ABvCvD	ABvC	ABvC	AvC	AvCvD	BvC	BvCvD	AvC	BvC	ABvC	
ABvD	T	AvD	BvD	ABvCvD	ABvD	ABvD	ABvD	AvCvD	AvD	BvCvD	BvD	AvD	BvD	ABvCvD	ABvD
ABvCD	T	AvCD	BvCD	ABvC	ABvD	ABvCD	ABvCD	AvC	AvD	BvC	BvD	AvCD	BvCD	ABvC	ABvD ABvCD

Chart II. Apex Gate performs A' v C' (or A' v D' or B' v C' or B' v D')

C'D'	T	F	A	B	C	D	AB	CD	AvC	AvD
AC	T	AC	A	BvAC	C	ACvD	A(BvC)	C(AvD)	AvC	AvD
AD	T	AD	A	BvAD	CvAD	D	A(BvD)	D(AvC)	AvC	AvD
ACD	T	ACD	A	BvACD	C	D	A(BvCD)	CD	AvC	AvD
A(BvC)	T	A(BvC)	A	BvAC	CvAB	DvABvAC	A(BvC)	ABvACvCD	AvC	AvD
A(BvD)	T	A(BvD)	A	BvAD	CvABvAD	ABvD	A(BvD)	ABvBDvCD	AvC	AvD
A(BvCD)	T	A(BvCD)	A	BvCD	CvAB	ABvD	A(BvCD)	ABvCD	AvC	AvD
BC	T	BC	AvBC	B	C	BCvD	A(BvC)	C(BvD)	AvC	AvDvBC
BD	T	BD	AvBD	B	CvBD	D	B(AvD)	D(BvC)	AvCvBD	AvD
BCD	T	BCD	AvBCD	B	C	D	B(AvCD)	CD	AvC	AvD
B(AvC)	T	B(AvC)	AvBC	B	CvAB	DvABvAC	B(AvC)	ABvBCvCD	AvC	AvDvBC
B(AvD)	T	B(AvD)	AvBD	B	CvABvBD	DvAB	B(AvD)	ABvBDvCD	AvCvBD	AvD
B(AvCD)	T	B(AvCD)	AvCD	B	CvAB	DvAB	B(AvCD)	ABvCD	AvC	AvD
ABC	T	ABC	A	B	C	ABCvD	AB	C(ABvD)	AvC	AvD
C(AvD)	T	C(AvD)	AvCD	BvACvCD	C	DvAC	ABvACvCD	C(AvD)	AvC	AvD
C(BvD)	T	C(BvD)	AvBCvBD	BvCD	C	DvBC	ABvBCvCD	C(BvD)	AvC	AvDvBC
C(ABvD)	T	C(ABvD)	AvCD	BvCD	C	DvABC	ABvCD	C(ABvD)	AvC	AvD
ABD	T	ABD	A	B	CvABD	D	AB	D(ABvC)	AvC	AvD
D(AvC)	T	D(AvC)	AvCD	BvADvCD	CvAD	D	ABvADvDC	D(AvC)	AvC	AvD
D(BvC)	T	D(BvC)	AvBDvCD	BvCD	CvBD	D	ABvBDvCD	D(BvC)	AvCvBD	AvD
D(ABvC)	T	D(ABvC)	AvCD	BvCD	CvABD	D	ABvCD	D(ABvC)	AvC	AvD
ABCD	T	ABCD	A	B	C	D	AB	CD	AvC	AvD
ABvADvBCvBD	T	ABvADvBCvBD	AvBCvBD	BvAD	CvABvADvBD	DvABvBC	ABvADvBCvBD	ABvADvBCvBDvCD	AvCvBD	AvDvBC
ABvBCvCD	T	ABvBCvCD	AvBDvCD	BvCD	CvAB	DvABvBC	ABvBCvCD	ABvBCvCD	AvC	AvDvBC
ABvADvCD	T	ABvADvCD	AvCD	BvADvCD	CvABvAD	ABvD	ABvADvCD	ABvADvCD	AvCv	AvD
ABvACvBDvCD	T	ABvACvBDvCD	AvBDvCD	BvACvCD	CvABvBD	DvABvAC	ABvACvBDvCD	ABvACvBDvCD	AvCvDB	AvD
ABvBCvCD	T	ABvBCvCD	AvBCvCD	BvCD	CvAB	DvABvBC	ABvBCvCD	ABvBCvCD	AvC	AvDvBC
ABvACvCD	T	ABvACvCD	AvCD	BvACvCD	CvAB	DvABvAC	ABvACvCD	ABvACvCD	AvC	AvD

Chart III. Part 1 apex gate performs A' v C'D'

	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
AC	BvC	BvDvAC	AvCD	BvACvCD	ABvC	ABvACvD	ABvACvCD
AD	ADvBvC	BvD	AvCD	BvADvCD	CvABvAD	ABvD	ABvADvCD
ACD	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
A(BvC)	BvC	BvDvAC	AvCD	BvACvCD	ABvC	ABvACvD	ABvACvCD
A(BvD)	BvCvAD	BvD	AvCD	BvADvCD	ABvADvC	ABvD	ABvADvCD
A(BvCD)	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
BC	BvC	BvD	AvBCvCD	BvCD	ABvC	ABvBCvD	ABvBCvCD
BD	BvC	BvD	AvBDvCD	BvCD	ABvBDvC	ABvD	ABvBDvCD
BCD	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
B(AvC)	BvCvAB	BvD	AvBCvCD	BvCD	ABvC	ABvBCvD	ABvBCvCD
B(AvD)	BvC	BvD	AvBDvCD	BvCD	ABvBDvC	ABvD	ABvBDvCD
ABC	BvC	BvD	AvBD	BvCD	ABvC	ABvD	ABvCD
C(AvD)	BvC	BvDvAC	AvCD	BvACvCD	ABvC	ABvACvD	ABvACvCD
C(AvD)	BvC	BvD	AvCvCD	BvCD	ABvC	ABvBCvD	ABvBCvCD
C(ABvD)	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
ABD	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
D(AvC)	BvCvAD	BvD	AvCD	BvADvCD	ABvADvC	ABvD	ABvADvCD
D(BvC)	BvC	BvD	AvBDvCD	BvCD	ABvBDvC	ABvD	ADvBDvCD
D(ABvC)	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
ABCD	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD
ABvADvBCvBD	BvCvAD	BvD	AvBCvBDvCD	BvADvCD	CvABvADvBD	ABvBCvD	ABvADvBCvBDvCD
ABvBCvCD	BvC	BvD	AvBCvCD	BvCD	ABvC	ABvBCvD	ABvBCvCD
ABvADvCD	BvCvAD	BvD	AvCD	BvADvCD	ABvADvC	ABvD	ABvADvCD
ABvACvBDvCD	BvC	BvDvAC	AvBDvCD	BvACvCD	ABvBDvC	ABvACvD	ABvACvBDvCD
ABvBCvCD	BvC	BvD	AvBCvCD	BvCD	ABvC	ABvBCvD	ABvBCvCD
ABvACvCD	BvC	BvDvAC	AvCD	BvACvCD	ABvC	ABvACvD	ABvACvCD
B(AvCD)	BvC	BvD	AvCD	BvCD	ABvC	ABvD	ABvCD

Chart III. Part 2 apex gate performs A' v C'D'

$$(A(BvCD)) = (B(AvCD)) = C(ABvD) = D(ABvC) = 2[(A)(BvCD) + (A)(ABvCD)] = .0094$$

$$(ABC) = (ABD) = (ACD) = (BCD) = 2(AB)(C) = .0012$$

$$(ABCD) = 2(AB)(CD) = .011$$

$$(F) = 2F(1-F) + F^2 = .02$$

The next step in our calculation is to compute the probability of the function ABvCD resulting from the recursion. As before we will enumerate around the apex function.

Apex function	Combinations yielding ABvCD	p{ ABvCD|Apex function}· p{Apex function}
A'	(ABvCD)	4(.008)(.53) = .017
A'vC'	$(ABvCD)^2$ + 2(AB)(CD) + 2(ABvCD)[(F) + (CD)]	4(.007)(.464) = .013
A' B'	$(ABvCD)^2$ + 2(ABvCD)[(T) + 4(AvC) + 4(AvCD)] + 4(AvCD)(BvCD)	2(.073)(.64) = .094
A'vB' C'	Note that an asterisk indicates that the member results from conjunct term and its value is taken from the preceding table $4(AB)(C(AB\overset{*}{v}CD)) + 4(ABvCD)(A\overset{*}{C}D) + 4(A\overset{*}{B}vCD)(A(B\overset{*}{v}CD)) + (AB\overset{*}{v}CD)(AB\overset{*}{C}D) + (AB\overset{*}{v}CD)(AB\overset{*}{v}CD) + 2(AB)(C\overset{*}{D}) + 2[(ABvCD)(A\overset{*}{B}) + (AB)(AB\overset{*}{v}CD)] + (ABvCD)(F) + (\overset{*}{F})(ABvCD)$	4(.059)(.630) = .149
A'B' vC' D'	Here all terms which appear result from a conjunction. $4(AB)(ABvCD) + 2(F)(ABvCD) + (ABvCD)^2 + 2(ABvCD)(F) + 8(AB)(C(ABvD)) + 8(ABC)(ABvCD) + 8(A(BvCD))(ABvCD) + 2(ABvCD)(ABCD) + 8(A(BvCD))(C(ABvD))$.	(.53)(.877) = .465

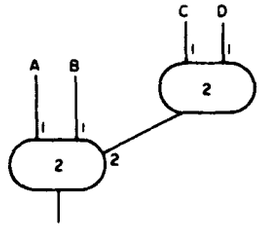
Therefore p{ABvCD} = .017 + .013 + .094 + .149 + .465 = .738 which is greater than .53 which is the p{ABvCD} in the basic gate. But this improvement has been bought at a cost of 5 times the amount of equipment in a nonredundant gate.

If we consider the second configuration described earlier in the report, where we applied recursion directly to each 2-input gate which composed the basic network, then one level of recursion (shown) costs only 3 times the amount of equipment in a basic gate. In an earlier report (Maitra) the probability of successful operation of a first level triangle of 2-input rectifier gates with p{rectifier failure} = 0.1 has been calculated. It is 0.940. Successful operation of the function ABvCD requires that all three triangles operate properly. This occurs with probability $(.940)^3 = .830$ which is greater than .738 and has been obtained at a cost only $^3/5$ as great. This preliminary study, then, appears to indicate that the simple basic gate is the more promising level at which to apply triangular recursion.

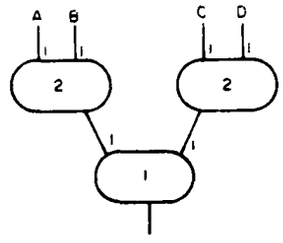
The Threshold Logic Realization

In addition to a realization of the function ABvCD using rectifier gates we attempted a realization using threshold gates. As in our previous report (Special Scientific Report No. 1) it was assumed that errors in operation resulted from variation in the bias.

The first threshold realization we considered was:



In this case recursion yielded no stability under threshold variation. The upper gate can be made more reliable by recursion but the lower gate cannot. So we attempted to use the same network as before to synthesize the function:



Again our assumptions are that errors in operation result from variations in the bias. These variations are represented by a normally distributed noise signal superimposed on the bias signal and centered at the appropriate level. Equally weighted two input threshold gates can realize only 4 positive Boolean functions: the constant T (or 1), Disjunction (or), Conjunction (and), the constant F (or 0).

The tables for the resultant function again taken for each apex function follow

A,B \ C,D	T	D	C	F	
T	T	T	T	T	
D	T	AvBvCvD	AvBvCD	AvB	Apex function Disjunction
C	T	ABvCvD	ABvCD	AB	
F	T	CvD	CD	F	

A,B \ C,D	T	D	C	F	
T	T	CvD	CD	F	
D	AvB	(AvB)(CvD)	(AvB)CD	F	Apex function Conjunction
C	AB	(AB)(CvD)	ABCD	F	
F	F	F	F	F	

The probabilities for the basic gate can be computed from the tables. The notation used below is: M_N is the probability that a gate which is intended to be an N gate is an M gate; thus, C_D is the probability that a disjunction (or) gate will act as a conjunction (and) gate

$$p\{T\} = T_D + C_D[T_C^2] + D_D[2T_C - T_C^2]$$

$$p\{F\} = F_D + C_D[2F_C - F_C^2] + D_D[F_C^2]$$

$$p\{ABvCD\} = D_D C_C^2$$

$$p\{AB\} = p\{CD\} = D_D C_C F_C + C_D C_C T_C$$

$$p\{AvB\} = p\{CvD\} = D_D D_C F_C + C_D D_C T_C$$

$$p\{ABvCvD\} = p\{AvBvCD\} = D_D C_C D_C$$

$$p\{AB(CvD)\} = p\{(AvB)CD\} = D_C C_D D_D$$

$$p\{AvBvCvD\} = D_D D_C^2$$

$$p\{ABCD\} = C_D C_C^2$$

$$p\{ACvADvBCvBD\} = D_C C_D^2$$

If we consider $\sigma = 0.50$ in our normal distribution, then

$$C_C = D_D = .6826900$$

$$D_C = C_D = .1573050$$

$$T_D = F_C = .1586550$$

$$T_C = F_D = .0013500$$

Inserting these values into the equations on the previous page, we get:

$$p\{T\} = .1604961$$
$$p\{F\} = .06052949$$
$$p\{ABvCD\} = .3181783$$
$$p\{AB\} = p\{CD\} = .0741341$$
$$p\{AvB\} = p\{CvD\} = .0170767$$
$$p\{ABvCvD\} = p\{AvBvCD\} = .073314$$
$$p\{AB(CvD)\} = p\{(AvB)CD\} = .01689$$
$$p\{ABCD\} = .073314$$
$$p\{AvBvCvD\} = .01689$$
$$p\{(AvB)(CvD)\} = .01689$$

We are recurring the entire network as in the first case and we shall develop our expression around the functions of the apex gate. Again, we make use of charts to assist our calculation and we use the same abbreviations as before $(x) \equiv p\{x\}$.

T											
F	F										
ABvCD	F	ABvCD									
AB	F	AB	AB								
CD	F	CD	ABCD	CD							
AvB	F	ABvACDvBCD	AB	(AvB)CD	AvB						
CvD	F	ABCvABDvCD	AB(CvD)	CD	ACvADv BCvBD	CvD					
ABvCvD	F	ABvCD	AB	CD	ABvACvAD vBCvBD	CvD	ABvCvD				
AvBvCD	F	ABvCD	AB	CD	AvB	ACvADvBC vBDvCD	ABvACvAD vBCvBDvCD	AvBvCD			
AB(CvD)	F	AB(CvD)	AB(CvD)	ABCD	AB(CvD)	AB(CvD)	AB(CvD)	AB(CvD)	AB(CvD)		
(AvB)CD	F	(AvB)CD	ABCD	(AvB)CD	(AvB)CD	(AvB)CD	(AvB)CD	(AvB)CD	ABCD	(AvB)CD	
ABCD	F	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD
AvBvCvD	F	ABvCD	AB	CD	AvB	CvD	ABvCvD	AvBvCD	AB(CvD)	(AvB)CD	ABCD
(AvB)(CvD)	F	ABCvABD vACDvBCD	AB(CvD)	(AvB)CD	(AvB)(CvD)	(AvB) (vD)	(AvB) (CvD)	(AvB) (CvD)	AB(CvD)	(AvB)CD	ABCD
ABvACvAD vBCvBDvCD	F	ABvCD	AB	CD	ABvACvAD vBCvBD	ACvADvBC vBDvCD	ABvACvAD vBCvBDvCD	ABvACvAD vBCvBDvCD	AB(CvD)	(AvB)CD	ABCD
ABvACD vBCD	F	ABvACD vBCD	AB	(AvB)CD	ABvACD vBCD	ABCvABD vACDvBCD	ABvACD vBCD	ABvACD vBCD	AB(CvD)	(AvB)CD	ABCD
ABvACvAD vBCvBD	F	ABvACD vBCD	AB	(AvB)CD	ABvACvAD vBCvBD	(AvB) (CvD)	ABvACvAD vBCvBD	ABvACvAD vBCvBD	AB(CvD)	(AvB)CD	ABCD
ABCvABDvCD	F	ABCvABD vCD	AB(CvD)	CD	ABCvABD vACDvBCD	ABCvABD vCD	ABCvABD vCD	ABCvABD vCD	AB(CvD)	(AvB)CD	ABCD
ACvADvBC vBDvCD	F	ABCvABD vCD	AB(CvD)	CD	(AvB)(CvD)	ACvADvBC vBDvCD	ACvADvBC vBDvCD	ACvADvBC vBDvCD	AB(CvD)	(AvB)CD	ABCD
ABCvABD vACDvBCD	F	ABCvABD vACDvBCD	AB(CvD)	(AvB)CD	ABCvABD vACDvBCD	ABCvABD vACDvBCD	ABCvABD vACDvBCD	ABCvABD vACDvBCD	AB(CvD)	(AvB)CD	ABCD

Apex Gate A' B' Part I

AvBvCD

(AvB)(CvD) (AvB)(CvD)

ABvACvAD vBCvBDvCD (AvB)(CvD) ABvACvAD vBCvBDvCD

ABvACD vBCD ABCvACD vABDvBCD ABvACD vBCD ABvACD vBCD

ABvACvAD vBCvBD (AvB)(CvD) ABvACvAD vBCvBD ABvACD vBCD ABvACvAD vBCvBD

ABCvABD vCD ABCvABD vCD ABCvABD vACDvBCD ABCvABD vACDvBCD ABCvABD vCD

ACvADvBC vBDvCD (AvB)(CvB) ACvADvBC vBDvCD ABCvABD vACDvBCD (AvB)(CvD) ABCvABD vCD ACvADvBC vBDvCD

ABCvABDv ACDvBCD ABCvABDv ACDvBCD ABCvABDv ACDvBCD ABCvABDv ACDvBCD ABCvABDv ACDvBCD ABCvABDv ACDvBCD ABCvABDv ACDvBCD

Apex Gate A' B' Part II

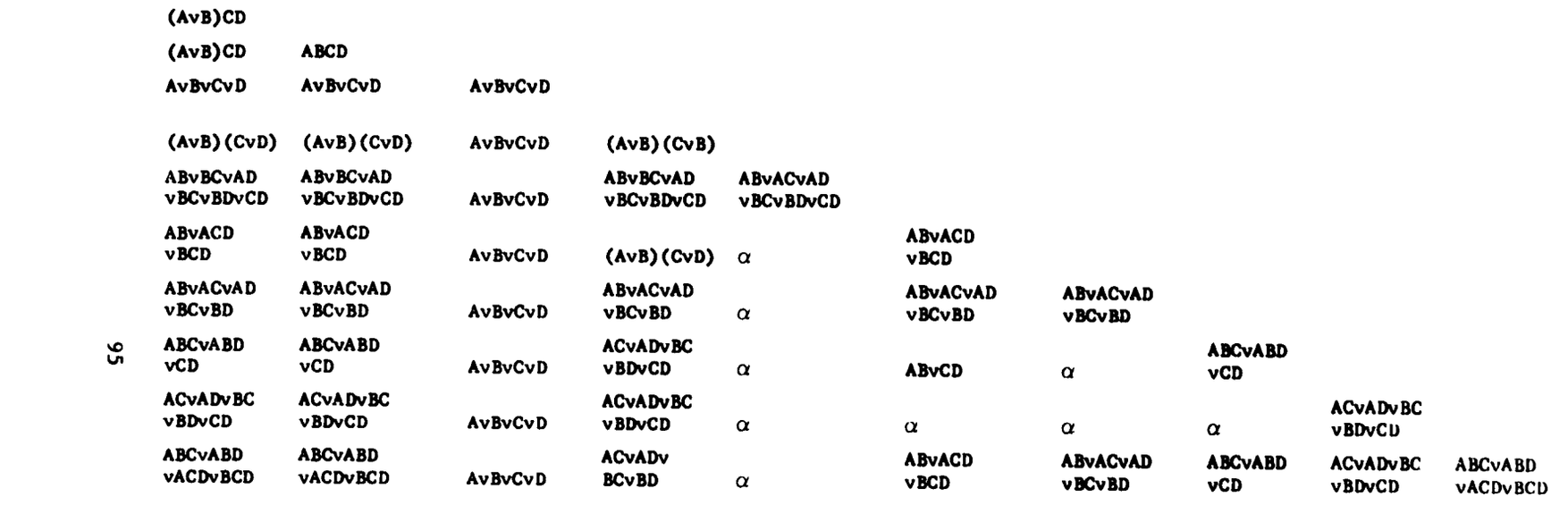
	A'	B'						
T	α	ACvADvBC vBDvCD	ABvACvAD BCvBD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
AvBvCD	α	ACvADv BCvBDvCD	ABvACvADv BCvBD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
AvB	ABvACvAD BCvBD	ACvADvBC vBD	ABvACvBC vBDvAD	ACvAD vBCvBD	ACDvBCD	ABCvABD	ABvACD vBCD	ACDvBCD ABCvABD
ABvCvD	α	ACvADvBC vBDvCD	ABvACvBC vBDvAD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
AvBvCvD	α	ACvADvBC vBDvCD	ABvACvBC vBDvAD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
AB	AB	ABCvABD	AB	ABCvABD	ABCD	ABCvABD	AB	ABCvABD
CvD	ACvADvBC vBDvCD	ABCvABD	ACvBCvBD vAD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABCvABD vACDvBCD	CDvABC vABD
CD	CD	CD	ACDvBCD	ACDvBCD	ACDvBCD	ABCD	ACDvBCD	CD
F	F	F	F	F	F	F	F	F
ABvCD	ABvCD	ABCvABD vCD	ABvACD vBCD	ABCvABD vBCD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD

Part III

F									
T	T								
ABvCD	T	ABvCD							
AB	T	ABvCD	AB						
CD	T	ABvCD	ABvCD	CD					
AvB	T	AvBvCD	AvB	AvBvCD	AvB				
CvD	T	ABvCvD	ABvCvD	CvD	AvBvCvD	CvD			
ABvCvD	T	ABvCvD	ABvCvD	ABvCvD	AvBvCvD	ABvCvD	ABvCvD		
AvBvCD	T	AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCvD	AvBvCvD	AvBvCD	
AB(CvD)	T	ABvCD	AB	ABCvABD vCD	AvB	CvD	ABvCvD	AvBvCD	AB(CvD)
(AvB)CD	T	ABvCD	ABvACD vBCD	CD	AvB	CvD	ABvCvD	AvBvCD	ABCvABD vACDvBCD
ABCD	T	ABvCD	AB	CD	AvB	CvD	ABvCvD	AvBvCD	AB(CvD)
AvBvCvD	T	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD
(AvB)(CvD)	T	ABvACvAD vBCvBDvCD	ABvACvAD vBCvBD	ACvADvBC vBDvCD	AvB	CvD	ABvCvD	AvBvCD	(AvB)(CvD)
ABvBCvAD vBCvBDvCD	T	ABvBCvAD vBCvBDvCD	ABvBCvAD vBCvBDvCD	ABvBCvAD vBCvBDvCD	AvBvCD	ABvCvD	ABvCvD	AvBvCD	ABvBCvAD vBCvBDvCD
ABvACD vBCD	T	ABvCD	ABvACD vBCD	ABvCD	AvB	ABvCvD	ABvCvD	AvBvCD	ABvACD vBCD
ABvACvAD vBCvBD	T	α	ABvACvAD vBCvBD	α	AvB	ABvCvD	ABvCvD	AvBvCD	ABvACvAD vBCvBD
ABCvABD vCD	T	ABvCD	ABvCD	ABCvABD vCD	AvBvCD	CvD	ABvCvD	AvBvCD	ABCvABD vCD
ACvADvBC vBDvCD	T	α	α	ACvADvBC vBDvCD	AvBvCD	CvD	ABvCvD	AvBvCD	ACvADvBC vBDvCD
ABCvABD vACDvBCD	T	ABvCD	ABvACD vBCD	ABCvABD vCD	AvB	CvD	ABvCvD	AvBvCD	ABCvABD vACDvBCD

α = ABvBCvADvBCvBDvCD

Apex Gate A' v B' Part I



Apex Gate A' v B' Part II

A' v B'

T	T	T	T	T	T	T	T	T
AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCD	AvBvCD
AvB	AvBvCD	AvBvCD	AvB	AvB	AvB	AvB	AvB	AvBvCD
ABvCvD	ABvCvD	ABvCvD	ABvCvD	ABvCvD	ABvCvD	ABvCvD	ABvCvD	ABvCvD
AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD
AB	α	α	ABvACvBC vADvBD	ABvACvBC vADvBD	ABvACD vBCD	AB	ABvACD vBCD	ABvCD
CvD	ABvCvD	CvD	ABvCvD	CvD	CvD	CvD	ABvCvD	CvD
CD	α	ACvADvBC vBDvCD	α	ACvADvBC vBDvCD	CD	ABCvABD vCD	ABvCD	CDvABCvABD
F	α	ACvADvBC vBDvCD	ABvACvBC vADvBD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD
ABvCD	α	α	α	α	ABvCD	ABvCD	ABvCD	ABvCD
ABCD	α	ACvADvBC vBDvCD	ABvACvBC vADvCD	ACvADvBC vBD	ACDvBCD	ABCvABD	ABvACD vBCD	CDvABC vABD

Part III

A' B'

ABvACvAD vBCvBDvCD	ABvACvAD vBCvBDvCD							
ACvADvBC vBDvCD	ACvADvBC vBDvCD	ACvADvBC vBDvCD						
ABvACvBC vADvBD	ABvACvBC vADvBD	ACvBCvAD vBD	ABvACvBC vBDvAD					
ACvADvBC vBD	ACvADvBC vBD	ACvADvBC vBD	ACvADvBC vBD	ACvAD vBCvBD				
ACDvBCD	ACDvBCD	ACDvBCD	ACDvBCD	ACDvBCD	ACDvBCD			
ABCvABD	ABCvABD	ABCvABD	ABCvABD	ABCvABD	ABCD	ABCvABD		
ABvACD vBCD	ABvACD vBCD	ACDvBCDv ABCvABD	ABvACD vBCD	ACDvBCD vABCvABD	ACDvBCD	ABCvABD	ABvACD vBCD	
CDvABC vABD	CDvABC vABD	ABCvABD vCD	ABCvABD vACDvBCD	ABCvABD vACDvBCD	ACDvBCD	ABCvABD	ABCvABD vACDvBCD	CDvABCvABD

A' v B'

α	α							
ACvADvBC vBDvCD	α	ACvADvBC vBDvCD						
ABvACvBC vADvBD	α	α	ABvACvBC vADvBD					
ACvADvBC vBD	α	ACvADvBC vBDvCD	ABvACvBC vADvBD	ACvADvBC vBD				
ACDvBCD	α	ACvADvBC vBDvCD	ABvACvBC vADvBD	ACvADvBC vBD	ACDvBCD			
ABCvABD	α	ACvADvBC vBDvCD	ABvACvBC vADvBD	ACvADvBC vBD	ABCvABD vACDvBCD	ABCvABD		
ABvACD vBCD	α	α	ABvACvBC vADvBD	ACvADvBC vBD	ABvACD vBCD	ABvACD vBCD	ABvACD vBCD	
CDvABCv ABD	α	ACvADvBC vBDvCD	α	ACvADvBC vBD	CDvABC vABD	CDvABC vABD	ABvCD	CDvABC vABD

α = ABvACvADvBCvBDvCD

T	AB(CvD)	(AvB)CD	(AvB)(CvD)
F	F	F	F
ABvCD	AB(CvD)	(AvB)CD	ABCvABD vACDvBCD
AB	AB(CvD)	ABCD	AB(CvD)
CD	ABCD	(AvB)CD	(AvB)CD
AvB	AB(CvD)	(AvB)CD	(AvB)(CvD)
CvD	AB(CvD)	(AvB)(CD)	(AvB)(CvD)
ABvCvD	AB(CvD)	(AvB)CD	(AvB)(CvD)
AvBvCD	AB(CvD)	(AvB)CD	(AvB)(CvD)
AB(CvD)	AB(CvD)	ABCD	(AB)(CvD)
(AvB)CD	ABCD	(AvB)CD	(AvB)CD
ABCD	ABCD	ABCD	ABCD
AvBvCvD	AB(CvD)	(AvB)CD	(AvB)(CvD)
(AvB)(CvD)	AB(CvD)	(AvB)CD	(AvB)(CvD)

Additional A' B'

F	AB(CvD)	(AvB)(CD)	(AvB)(CvD)
T	T	T	T
ABvCD	ABvCD	ABvCD	α
AB	AB	ABvACD vBCD	ABvACvAD vBCvBD
CD	ABCvABD vCD	CD	ABvACvAD vBDvCD
AvB	AvB	AvB	AvB
CvD	CvD	CvD	CvD
ABvCvD	ABvCvD	ABvCvD	ABvCvD
AvBvCD	AvBvCD	AvBvCD	AvBvCD
AB(CvD)	AB(CvD)	ABCvABD vACDvBCD	(AvB)(CvD)
(AvB)CD	ABCvABD vACDvBCD	(AvB)(CD)	(AvB)(CvD)
ABCD	AB(CvD)	(AvB)(CD)	(AvB)(CvD)
AvBvCvD	AvBvCvD	AvBvCvD	AvBvCvD
(AvB)(CvD)	(AvB)(CvD)	(AvB)(CvD)	(AvB)(CvD)

Additional A' v B'

Part V

Combinations Yielding ABvCD

Apex gate

2(AB) $2(ABvCD)T + 2(ABvCD)(AvBvCD) + 2(ABvCD)(ABvCvD) + 2(ABvCD)(AvBvCvD) + (ABvCD)^2$

This expression (above) will be denoted by α and the corresponding expression when the apex function is AvB will be denoted by β. In addition the probability of a function F resulting from either an AB apex or an AvB apex will be denoted by $(F)_\alpha$ or $(F)_\beta$ respectively.

2(AvB) $2(ABvCD)(F) + 2(AB)(CD) + 2(ABvCD)(AB) + 2(ABvCD)(CD) + (ABvCD)^2 + 2(ABvCD)(ABCD) + 2(ABCD)(AB(CvD)) + 2(ABvCD)((AvB)CD)$

ABvCD $2(\alpha)(F_\alpha) + 2(AB_\alpha)(CD_\alpha) + 2(\alpha)(AB_\alpha) + 2(\alpha)(CD_\alpha) + (\alpha)^2 + 2(\alpha)(ABCD_\alpha) + 2(\alpha)([AB(CvD)]_\alpha) + 2(\alpha)([(AvB)CD]_\alpha) + 2(ABvACvADvBCvBDvCD)_\alpha$

ABvCvD $(\alpha)(F_\beta) + (\alpha)(\beta) + (\alpha)(AB_\beta) + (\alpha)(CD_\beta) + (\beta)(F_\alpha) + (\beta)(AB_\alpha) + (\beta)(CD_\alpha) + (CD_\alpha)(AB_\beta) + ([AB(CvD)]_\alpha)(\beta) + ([AvB)CD]_\alpha)(\beta) + (CD_\beta)(AB_\alpha) + (\alpha)([AB(CvD)]_\beta) + (\alpha)([(AvB)CD]_\beta) + (ABCD_\alpha)(\beta) + ([ABvACDvBCD]_\alpha)(\beta) + ([ABvACDvBCD]_\alpha)(CD_\beta) + (ABCD_\beta)(\alpha) + (\alpha)([ABvACDvBCD]_\beta) + ([ABvACDvBCD]_\beta)(CD_\alpha) + ([ABCvABDvCD]_\alpha)(\beta) + ([ABCvABDvCD]_\alpha)(AB_\beta) + ([ABCvABDvCD]_\beta)(\alpha) + ([ABCvABDvCD]_\beta)(AB_\alpha) + ([ABCvABDvCD]_\alpha)([ABvACDvBCD]_\beta) + (\alpha)([ABCvABDvACDvBCD]_\beta) + ([ABCvABDvCD]_\beta)([ABvACDvBCD]_\beta) + (\beta)([ABCvABDvACDvBCD]_\alpha)$

AB(CvD) $(T_\alpha)(\beta) + (\alpha)(\beta) + ([ABvCvD]_\alpha)(\beta) + (\alpha)(T_\beta) + (\alpha)([ABvCvD]_\beta) + ([AvBvCD]_\alpha)(\beta) + ([AvBvCvD]_\alpha)(\beta) + ([AvBvCD]_\beta) + (\alpha)([AvBvCvD]_\alpha) + ([ABvACvADvBCvBDvCD]_\alpha)(\beta) + (\alpha)([ABvACvADvBCvBDvCD]_\beta)$

ABCD $2(\alpha)(T_\alpha) + (\alpha)^2 + 2([ABvCvD]_\alpha)(\alpha) + 2([AvBvCD]_\alpha)(\alpha) + 2([AvBvCvD]_\alpha)(\alpha) + 2([ABvACvADvBCvBDvCD]_\alpha)(\alpha)$

(AvB)(CvD) $2(\beta)(T_\beta) + \beta^2 + 2([ABvCD]_\beta)(\beta) + 2([AvBvCD]_\beta)(\beta) + 2([AvBvCvD]_\beta)(\beta) + 2([ABvACvADvBCvBDvCD]_\beta)(\beta)$

$$(AvB)v(CvD)\ \ 2(\beta)(F_\beta) + (\beta)^2 + 2(\beta)(AB_\beta) + 2(\beta)(CD_\beta) + 2(AB_\beta)(CD_\beta)$$
$$+ 2([AB(CvD)]_\beta)(\beta) + 2([AvB)CD]_\beta)(\beta) + 2(ABCD_\beta)(\beta)$$
$$+ 2([ABvACDvBCD]_\beta)(\beta) + 2([ABvACDvBCD]_\beta)(CD_\beta) + 2([ABCvABDvCD]_\beta)(\beta)$$
$$+ 2([ABCvABDvCD]_\beta)(AB_\beta) + 2([ABCvABDvCD]_\beta)([ABvACDvBCD]_\beta)$$
$$+ 2(\beta)([ABCvABDvACDvBCD]_\beta)$$

$$F_\alpha = 2F-F^2$$

$$(AB)_\alpha = (CD)_\alpha = 2(AB)(ABvCD) + (AB)^2 + 2(AB)(AvB) + 2(AB)(T)$$
$$+ 2(AB)(AvBvCD) + 2(AB)(AvBvCvD)$$

$$(ABCD)_\alpha = 2(ABCD)(1-F) - (ABCD)^2$$

$$[AB(CvD)]_\alpha =$$
$$[(AvB)CD]_\alpha = 2(AB(CvD))(T) + 2(AB(CvD)(ABvCD) + 2(AB)(CvD)$$
$$+ 2(AB)(AB(CvD)) + 2(AB)((AvB)(CvD)) + 2((AB)(CvD))(AvB)$$
$$+ 2(AB(CvD))(CvD) + 2(AB(CvD))(ABvCvD)$$
$$+ 2(AB(CvD))(AvBvCD) + [AB(CvD)]^2$$

$$[ABvACvADvBCvBDvCD]_\alpha = 2(AvBvCD)(ABvCvD)$$

$$[ABvACDvBCD]_\alpha =$$
$$[ABCvABDvCD]_\alpha = 2(AvB)(ABvCD)$$

$$[ABCvABDvACDvBCD]_\alpha = 2((AvB)(CvD))(ABvCD)$$

$$T_\alpha = T^2$$

$$[AvBvCvD]_\alpha = 2(AvBvCvD)(T) + (AvBvCvD)^2$$

$$[ABvCvD]_\alpha =$$
$$[AvBvCD]_\alpha = 2(AvBvCD)(T) + (AvBvCD)^2$$

$$T_\beta = 2T - T^2$$

$$F_\beta = F^2$$

$$A\beta_\beta = CD_\beta = 2(AB)(F) + (AB)^2 + 2(AB)(AB(CvD)) + 2(AB)(ABCD)$$

$$[AB(CvD)]_\beta =$$
$$[(AvB)CD]_\beta = 2[(AB)(CvD)](F) + (AB(CvD))^2 + 2(AB(CvD))(ABCD)$$

$$[ABCD]_\beta = 2(ABCD)F + (ABCD)^2$$

$$[ABvACDvBCD]_\beta =$$
$$[ABCvABDvCD]_\beta = 2(AB(CvD))(CD)$$

$$[ABCvABDvACDvBCD]_\beta = 2((AvB)(CD))((AB)(CvD))$$

$$[AvBvCvD]_\beta = 2(AvBvCvD)(1-T) - (AvBvCvD)^2 + 2(AvB)(CvD)$$
$$+ 2(AvB)(ABvCvD) + 2(AvBvCD)(CvD) + 2(ABvCvD)(AvBvCD)$$

$$[ABvCvD]_\beta =$$
$$[AvBvCD]_\beta = 2(ABvCvD)(F) + 2(ABvCD)(CvD) + 2(ABvCD)(ABvCvD)$$
$$+ 2(AB)(CvD) + 2(ABvCvD)(AB) + 2(ABvCvD)(CD)$$
$$+ 2(ABvCvD)(CvD) + (ABvCvD)^2 + 2(ABvCvD)(AB(CvD))$$
$$+ 2(ABvCvD)(AvB)CD) + 2(ABvCvD)(ABCD)$$
$$+ 2(ABvCvD)((AvB)(CvD))$$

$$[ABvACvADvBCvBDvCD]_\beta = 2[(AvB)(CvD)][ABvCD]$$

Then plugging the values into these expressions get:

F_α	.11739516	α = .30742597
$[ABvACvADvBCvBDvCD]_\alpha$	.01074988	
$(AB)_\alpha = (CD)_\alpha$	.09237438	
$(ABCD)_\alpha$	.1323777	
$(ABvACDvBCD)_\alpha =$		
$(ABCvABDvCD)_\alpha$	.0471757	
$(ABCvABDvACDvBCD)_\alpha$	.01086687	
$(AvBvCvD)_\alpha$	.0057058	
$(AvBvCD)_\alpha = (ABvCvD)_\alpha$	.0057058	
T_α	.025758998	
$[AB(CvD)]_\alpha = [(AvB)CD]_\alpha$	.03095794	
F_β	.0036638	β = .3132488
$(ABvACvADvBCvBDvCD)_\beta$	.01074806	
$(AB)_\beta = (CD)_\beta$	.0278448	
$(ABCD)_\beta$	.0142649	

$(ABvACDvBCD)_\beta = (ABCvABDvCD)_\beta$	.0025042
$(ABCvABDvACDvBCD)_\beta$	.01074988
$(AvBvCvD)_\beta$	.04441408
$(AvBvCD)_\beta = (ABvCvD)_\beta$	.1164174
T_β	.2952332
$[AB(CvD)]_\beta = [(AvB)CD]_\beta$	.0048

and finally $p\{ABvCD\} = .245$ which is less than .318, the probability of correct operation of the nonredundant gate.

In addition the usual recursion of the basic gate yields no improvement in reliability, and it appears that in this case triangular recursion is of no value for improving the operation.

PART III. THE NOR GATE

Up to this point, we have not actually applied recursive triangulation to a complete set of Boolean functions - that is a set from which any Boolean function can be synthesized. The "nor" gate is such a function. Consider a rectifier transistor realization:

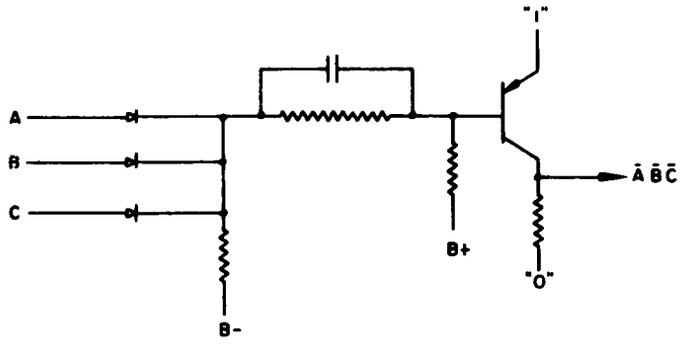
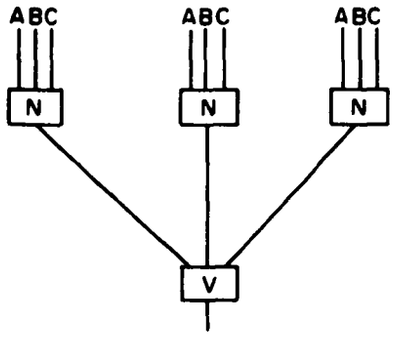


Fig. 8 Nor gate

If we consider only the active components failing then the distribution of functional states resulting from the component failures are $\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{A}\bar{B}$, $\bar{A}\bar{C}$, $\bar{B}\bar{C}$, $\bar{A}\bar{B}\bar{C}$, T, F. These are the same as in the case of the 3-input rectifier "and" gate except for the addition of the state F. If we apply regular recursion then 1 level of recursion can not yield the "nor" function and, in fact, from one level on the function computed will be positive. So we seek the best combining element for "nor" gates. We know from our studies of the rectifier gates that the "and" gate would be the ideal combining element were it not for the presence of the F. So we will study the effect of the F if we attempt to combine three "nor" gates into a first level recursive triangle.

We assume mixed technologies (more than one type of gate available). The object of the study is to determine how we could best improve the reliability of the "nor" gate. The technique used compares 3 combining elements -- a rectifier "or" gate, a rectifier "and" gate, and a majority gate.

A. Rectifier "or" Gate Combining Element



The distribution on the nor gate is again over $\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{A}\bar{B}$, $\bar{A}\bar{C}$, $\bar{B}\bar{C}$, $\bar{A}\bar{B}\bar{C}$, F, T, $p\{F\} = \frac{p}{10}$, using a rectifier "nor" gate as shown, with p = 0.2, and using the notation of the previous report:

$$Q = p\{\bar{A}\} = p\{\bar{B}\} = p\{\bar{C}\} = p^2(1-p)(1-\tfrac{p}{10}) = (.04)(.8)(.98) = .0314$$

$$R = p\{\bar{A}\bar{B}\} = p\{\bar{A}\bar{C}\} = p\{\bar{B}\bar{C}\} = p(1-p)^2(1-\tfrac{p}{10}) = (.2)(.64)(.98) = .125$$

$$T = p\{T\} = (p^3)(1-\tfrac{p}{10}) = (.08)(.98) = .00785$$

$$S = p\{\bar{A}\bar{B}\bar{C}\} = (1-p)^3\,(1-\tfrac{p}{10}) = (.512)(.98) = .504$$

$$p\{F\} = .02$$

The only way to get $\bar{A}\bar{B}\bar{C}$ from this combination is to have at least one of the input gates functioning properly and the diode fed by that gate also functioning properly.

$$p\{\bar{A}\bar{B}\bar{C}\} = 3(p^2)(1-p)\ (.504) + 3p(1-p)^2\ ((.504)^2 + 2(.504)(.02))$$
$$+ (1-p)^3\ ((.504) + 3(.504)^2\ (.02) + 3(.504)(.02)^2)$$
$$= 3(.032)(.504) + 3(.128)((.504)^2 + (.504)(.04))$$
$$+ .512\ ((.504)^3 + (.06)(.504)^2 + 3(504)(.02)^2)$$
$$= .0484$$

$p\{\bar{A}\bar{B}\bar{C}\} = .227$ no improvement; in fact, a decline.

Notice that if the combining gate is improved by triangular recursion to its maximum reliability p{or} = .991 that the probabilities of sensitivity to only 1 or only 2 inputs goes effectively to 0 and the only term which need be considered is the final term.

$p\{\tilde{A}\tilde{B}\tilde{C}\} = (.991)(.144) = .143$, still more of a decline.

In fact, a comparison of this calculation with the last shows that greater improvement results when the combining element is not operating as an "or" gate but rather when it is malfunctioning.

B. "And" Gate Combining Element

The recurrence relation is almost identical with the relation established for the "and" gate except that a factor should be added to ensure that no input function is F.

$$p\{\text{Nor}\} \geq p\{\text{first level and gate recursion}\} \cdot p\{\text{no input F}\}$$
$$= (.889)(.98)^3\,(.889)(.94) = .837 \text{ an improvement.}$$

In addition, suppose that the apex gate is improved by triangular recursion to be most reliable. Again p(AB), p(A) etc., go to 0, so that the only combinations left to yield "nor" are the following

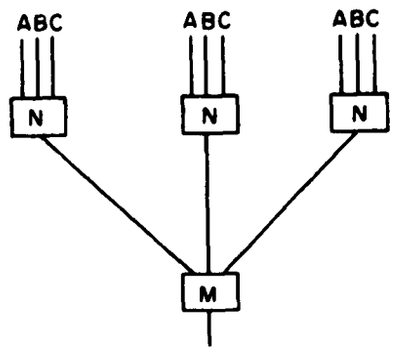
$$[S]_{\text{Apex}}\,[6Q^3 + 24R^3 + S^3 + 18QR^2 + 3T^2S + 3TS^2 + 63Q^2R + 63QR^2 + 27Q^2 + 3QS^2 + 27R^2S + 9RS^2 + 18QRT + 18QST + 18RST + 54QRS]$$

$S_{\text{Apex}} = .991$

Then, substituting:

$$[.991][6(.0314)^3 + 24(.125)^3 + (.504)^3 + 18(.0314)(.125)^2 + 3(.504)(.0785)^2 + 3(.504)^2\,(.0785) + 63(.0314)^2\,(.125) + 63(.0314)(.125)^2 + 27(.0314)^2\,(.504) + 9(.0314)(.504)^2 + 27(.125)^2\,(.504) + 9(.125)(.504)^2 + 18(.0314)(.125)(.0785) + 18(.0314)(.504)(.0785) + 18(.125)(.504)(.0785) + 54(.0314)(.125)(.504)] = .975, \text{ a decided improvement.}$$

C. Majority Gate Apex Gate



Since we are trying to demonstrate that the "and" gate is the superior combining element, we can only emphasize the point if we assume that we are working with a perfect majority gate. Now using the tables on the next two pages and expanding around one of the 3 input gates:

$$
\begin{aligned}
p[\bar{A}\bar{B}\bar{C}] = {} & F[6QR + 6R^2 + 2ST + S^2 + 2S(1-F-S)] \\
& + T[S^2 + 2SF] \\
& + 3Q[S^2 + 2RS + 2FS + 2RF] \\
& + 3R[S^2 + 4RS + 2QF + 4RF + 2R^2 + 2SF] \\
& + S[S^2 + 2S(1-S) + 2F(1-F) - 2SF + 2TS + 6R^2 + 6QR]
\end{aligned}
$$

F

$\bar{A}$	$\bar{A}$								
$\bar{B}$	$\bar{A}\bar{B}$	$\bar{B}$							
$\bar{C}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{C}$						
$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}$					
$\bar{A}\bar{C}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{C}$				
$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{B}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{B}\bar{C}$			
$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$		
T	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	T	
F	F	F	F	F	F	F	F	F	F

T

$\bar{A}$	$\bar{A}$								
$\bar{B}$	$\bar{A}\text{v}\bar{B}$	$\bar{B}$							
$\bar{C}$	$\bar{A}\text{v}\bar{C}$	$\bar{B}\text{v}\bar{C}$	$\bar{C}$						
$\bar{A}\bar{B}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\bar{B}$					
$\bar{A}\bar{C}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}(\bar{B}\text{v}\bar{C})$	$\bar{A}\bar{C}$				
$\bar{B}\bar{C}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{B}(\bar{A}\text{v}\bar{C})$	$\bar{C}(\bar{A}\text{v}\bar{B})$	$\bar{B}\bar{C}$			
$\bar{A}\bar{B}\bar{C}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$		
T	T	T	T	T	T	T	T	T	
F	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	T	F

This chart and the one following are for the case that the apex gate is a perfect majority gate. The two dimensions of each chart stand for two of the three input gates. The label on each section stands for the third input gate.

$\bar{A}$

$\bar{A}$	$\bar{A}$							
$\bar{B}$	$\bar{A}$	$\bar{B}$						
$\bar{C}$	$\bar{A}$	M	$\bar{C}$					
$\bar{A}\bar{B}$	$\bar{A}$	$\bar{A}\bar{B}$	$\bar{A}(\bar{B}v\bar{C})$	$\bar{A}\bar{B}$				
$\bar{A}\bar{C}$	$\bar{A}$	$\bar{A}(\bar{B}v\bar{C})$	$\bar{A}\bar{C}$	$\bar{A}(\bar{B}v\bar{C})$	$\bar{A}\bar{C}$			
$\bar{B}\bar{C}$	$\bar{A}$	$\bar{C}(\bar{A}v\bar{B})$	$\bar{C}(\bar{A}v\bar{B})$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$		
$\bar{A}\bar{B}\bar{C}$	$\bar{A}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	
T	$\bar{A}$	$\bar{A}v\bar{B}$	$\bar{A}v\bar{C}$	$\bar{A}$	$\bar{A}$	$\bar{A}v\bar{B}\bar{C}$	$\bar{A}$	T
F	$\bar{A}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}$ F

$\bar{A}\bar{B}$

$\bar{A}$	$\bar{A}$							
$\bar{B}$	$\bar{A}\bar{B}$	$\bar{B}$						
$\bar{C}$	$\bar{A}(\bar{B}v\bar{C})$	M	$\bar{C}$					
$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$				
$\bar{A}\bar{C}$	$\bar{A}(\bar{B}v\bar{C})$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$			
$\bar{B}\bar{C}$	$\bar{A}\bar{B}$	$\bar{B}(\bar{A}v\bar{C})$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{B}\bar{C}$		
$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	
T	$\bar{A}$	$\bar{B}$	$\bar{A}\bar{B}v\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}(\bar{B}v\bar{C})$	$\bar{B}(\bar{A}v\bar{C})$	$\bar{A}\bar{B}$	T
F	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}$ F

$\bar{A}\bar{B}\bar{C}$

$\bar{A}$	$\bar{A}$							
$\bar{B}$	$\bar{A}\bar{B}$	$\bar{B}$						
$\bar{C}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{C}$					
$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}$				
$\bar{A}\bar{C}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{C}$			
$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{B}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{B}\bar{C}$		
$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	
T	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\bar{B}$	$\bar{A}\bar{C}$	$\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	T
F	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$ F

$$[.02][6(.0314)(.125) + 6(.125)^2 + 2(.504)(.00785) + (.504)^2 + 2(.504)(.476)]$$
$$+ [.00785][(.504)^2 + 2(.504)(.02)]$$
$$+ [3(.0314)][(.504)^2 + 2(.125)(.504) + 2(.02)(.504) + 2(.125)(.02)]$$
$$+ [3(.125)][(.504)^2 + 4(.125)(.504) + 2(.0314)(.504) + 4(.125)(.02) + 2(.125)^2 + 2(.504)(.02)]$$
$$+ (.504)[(.504)^2 + 2(.504)(.496) + 2(.02)(.98) + 2(.00785)(.504) + 6(.125)^2 + 6(.0314)(.125)]$$
$$= .74355$$

This represents an improvement, but not nearly as great an improvement as with the "and" gate.

Suppose that we assume a much higher probability of F, occurring say at $p\{F\} = .2$

Then $Q = .0256$
$R = .1024$
$S = .4096$
$T = .0064$
$F = .2$

In the case of an "or" combining element, $p\{\bar{A}\bar{B}\bar{C}\} = .2490$ (>.227, the previous value), but still not an improvement over the basic gate.

In the case of an "and" combining element, $p\{\bar{A}\bar{B}\bar{C}\} = (.89)(.8)^3 = .455$, a decline over the previous value (.837) and only a small improvement over the basic gate reliability. So the effect of the "and" combining element appears to be quite sensitive to the value of $p\{F\}$.

Finally, the majority gate combining element yields $p\{\bar{A}\bar{B}\bar{C}\} = .675$, which represents a decline in the previous value but not as great as the decline which resulted with the "and" gate combining element. If instead

of the perfect majority gate we had assumed a majority gate similar to the type discussed under B. above then the values corresponding to p = .02 and p = .2 would be .687 and .631, respectively, a decline as before but actually a relatively insensitive reaction.

This last part has demonstrated that the choice of combining element is greatly influenced by the probability of the operating state, F. In the case that this probability is very small (the case more likely to actually occur) the "and" gate is the better combining element; as this probability increases the majority element begins to be more promising.

IV. ON THE USE OF FEEDBACK IN LOGICAL NETS TO PROVIDE SELF-CORRECTING CAPABILITIES

by C. V. Srinivasan

A. INTRODUCTION

The problem of developing techniques to construct reliable logical nets from inherently unreliable component devices is intrinsically of great value to computer designers. Considerable amount of creative talent has been directed in recent years toward discovering effective and practical solutions to the problem. Surely, there are many reasons for interest in this problem.

It is reasonable to expect that in the not too distant future it will be technologically possible to fabricate complex logical nets containing possibly millions of logical component units, each of which will be capable of operating at the rate of many millions of operations per second. Such units are likely to be of extremely small size. Also, it is not difficult for a computer engineer to conceive of a need for such large systems; the biological systems always provide the necessary inspiration. In such systems even a very small transient error probability of, say, one in a billion operations per component unit, is likely to cause millions of errors in the calculation every second. Hence, in order to make any meaningful computation at all, some form of automatic error correction is necessary. Further, in such systems having millions of components there will always exist some inevitable error in fabrication; and in the case of microminiature components, repair and maintenance will require replacements of basic modules, each such module being a fairly complex logical net. Therefore, to minimize the cost of such systems it is necessary to introduce some redundancy in their structure, so that a few failures will not necessitate a replacement of an entire module.

It is also true that the present-day computers are being frequently used to perform ambitious tasks, and oftentimes in such applications the cost of an error will be very high. So it is very important to construct nets of high reliability. In satellite systems, for example, the requirements on reliability are particularly severe, as in such systems error checking and maintenance by a human being is impossible, or, at least, is likely to be extremely costly. The above requirements have only introduced an urgency in the need for finding a solution to the problem of 'reliability'.

To define the concept of reliability of a logical net more precisely, it is necessary to understand the nature of errors and/or failures that are likely to occur in such nets and how one would wish to combat them.

A logical net, F, having m input lines and one output line, usually will perform a mapping of the values assumed by the input lines to the values that the output line can assume. Let us denote such a mapping by the logical function,

$$f_F^m (\underline{X}_i) = y_i \qquad (1)$$

where $\underline{X}_i$ is a possible valuation of the input lines and y_i is the corresponding output value. The net might have been designed to realize a desired logical function, f_D^m, and because of the possibility of erroneous operation of the logical components used in the net it is likely that f_F^m is sometimes different from f_D^m. If the errors in the net are transient, (we shall, hereafter, refer to this type of errors as Type I errors) then in a sequence of successive input-output mappings performed by the net (we shall, hereafter, refer to such a sequence of mappings as a <u>computation</u> done by the net) an input-output pair ($\underline{X}_i$, y_i) that does not satisfy the logical function, f_D^m, is likely to be followed by other pairs that do. However, every particular erroneous mapping (we shall, hereafter, refer to an individual mapping as a <u>calculation</u>) performed by the net was left uncorrected. To combat this type of error one would like to design nets in which the probability, p_i, of occurrence of an erroneous calculation is minimized. The smaller this probability is, the more reliable will be the calculations performed by the net.

In order to simplify calculations of error-probabilities in the case of transient errors, it will be necessary to make some assumptions on the probability distribution of error-occurrence as a function of time. To begin, we would like to assume that the probability of occurrence of a transient error would remain a constant for a sufficiently long period of operation of the net, as shown in Fig. 9.

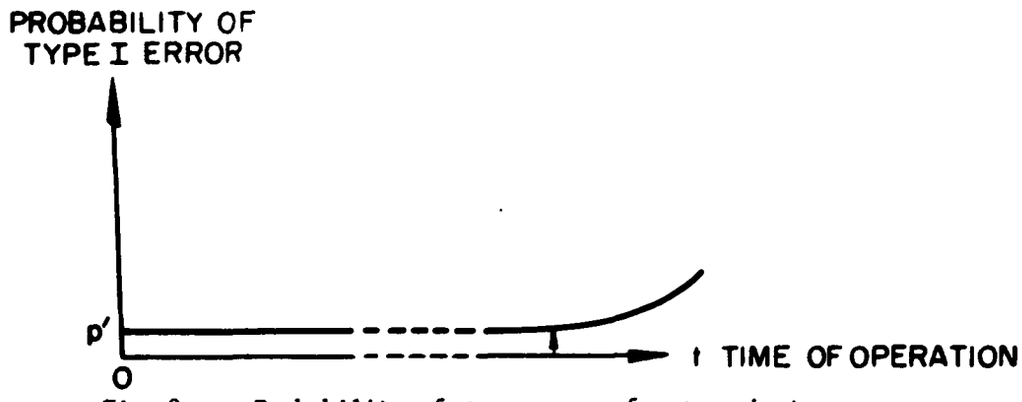


Fig. 9 Probability of occurrence of a transient error in a logical net, as a function of time.

As the period of operation of the net increases, due to an aging process, the net may deteriorate, thereby increasing the probability of occurrence of an error. To combat such deteriorating situations, one will have to develop some means of recognizing such deterioration and properly replace some components in the net. This calls for entirely different techniques of combating errors, and a discussion of this will be best done under a different error category.

Another assumption on transient error distribution will be that the probability of occurrence of such errors in successive calculations would be independent. This assumption is made primarily for the purpose of simplifying calculations to obtain error estimates. The scheme of error correction we shall employ does not depend on this assumption for successful operation. However, an estimate of improvement for the above case of independent probability distribution for successive calculations, will give us some idea about how effective the error-correcting had been. We may, therefore, write for Type I errors, that, for an input valuation, $\underline{X}_i$,

$$f_F^m(\underline{X}_i) = \begin{cases} f_D^m(\underline{X}_i) \text{ with probability } (1-p_i) \\ \bar{f}_D^m(\underline{X}_i) \text{ with probability } p_i. \end{cases} \quad (2)$$

in the case of nets where each input and output line assumes at a time only one of the two values 0 or 1 ($\bar{f}_D^m$ is the complement of f_D^m). Generally, the probability p_i could be a function of $\underline{X}_i$. In such a case one could safely choose the largest p_i as the first measure of unreliability of the net. Let $\underset{i}{\text{Max}}\ p_i$ be called p'.

However, if a permanent error (we shall, hereafter, call this Type II error) occurred in a net due to, say, the breakdown of a component in the net, then in a computation performed by the net, the outputs after the instant of occurrence of the error are likely to correspond to some other arbitrary function, f_E, or it may even just remain at a constant value. In the former case, if one knew what error occurred one could perhaps just use an appropriate additional calculation for error correction. In the latter case, of course, the net has to be replaced or repaired after the recognition of the error, and this requires that the computation be temporarily stopped. It is clear that to do any error correction, recognition of the error would be essential. To combat this type of error it is necessary either to construct nets with components of a very low probability of complete breakdown or to build into the net a capability to repair itself. Repair could be through error diagnosis and component replacement or else through a direct substitution of an alternate net in place of the one that had failed. Such error correction could be done also in anticipation of a failure due to, say, an aging process, referred to earlier.

If the probability of occurrence of a need for replacement, during a calculation is p" and p" remains a constant for a sufficiently long time, then one may choose p" as the second measure of unreliability of the net. Or, the expected value of the life of a net — the life being the period before the first repair — may be chosen as the second measure of reliability.

Two general techniques of interest and of some promise are at present known for designing logical nets having a p' (probability of occurrence of a Type I error) less than that of the component parts used in the nets. These two schemes also improve, to a certain extent, the second measure of reliability — viz. the life of the net. The first is von Neumann's[3] majority-vote-taker scheme and the second is the triangular recursive scheme[15,16] developed at this laboratory. In both cases errors in calculation are masked by the superfluous calculations that tend to reinforce preferentially the probability of generation of the correct output. Both these schemes do not offer the capability to detect and diagnose the

occurrence of errors. One might, therefore, say that these schemes are, in a sense, passive. The second scheme is even further restrictive, because it will not result in an improvement of reliability in all possible logical functions. ("not" function is a typical example where the triangular recursive scheme is not applicable.) However, for those functions for which it is applicable, the triangular scheme will yield greater improvement in reliability than the majority-vote-taker scheme. Thus, by the proper use of both the schemes, one may introduce a considerable improvement in the reliability of a logical net.

Logical nets that are designed to serve usefully for a long period of time should have some facility to combat the effects of deterioration of component units due to aging or other external interference. One would further like to have such a net take preventive measures to avoid a failure that may be anticipated. Such features will extend the expected life of a net. The passive schemes described above will not provide these capabilities. Some form of active error correction through error-detection and -repair is definitely called for to extend the useful life time of a logical net.

Such a self-correcting net would have to be basically a sequential net with a finite memory. The general problem of automatic error diagnosis and repair in logical nets with memory, seems to be a very difficult problem: at least, the presently available concepts appear to be inadequate to develop and analyze significant techniques.[18] In this report we present a simple scheme which, in a sense, is self-correcting. However, the correction is achieved through the substitution of a new unit in place of another that produced an erroneous calculation and not through error diagnosis and repair at the component level. In the context of this model, the problems that arise in the design of such nets are pointed out. This method of correction may be <u>uniformly</u> applied to all logical nets, as long as new substitutes are easily available. It is our contention that diagnosis and repair at the component level are essentially the properties of systems that possess the facility for practically unlimited growth — we have in mind systems that can produce practically unlimited quantities of the basic cells with which they are made.

Clearly, for a given component reliability, every improvement achieved in the reliability of a net, composed of the given components, has to be paid for in some form or other. In the passive schemes the designer is required to pay in terms of the increased number of components used in each net, and the consequent increased delay in operation. However, the designer does not have much control over the amount of time that a net will consume to perform an operation. Further, the amount of time delay will remain the same irrespective of whether an error occurred during a calculation or not. In the case of error detection and repair also the designer will have to pay in terms of greater number of components and greater time for computation. But, in this case, the time delay may be used more efficiently. For instance, the length of a computation will be large only in case an error occurred; otherwise the delay will be small. Also, the delay time may be used for error-diagnosis and error-signalling. As the component units deteriorate in a large logical system of self-correcting subnets, reliability will not decrease. However, more time will be needed for computation.

B. THE DESCRIPTION OF THE MODEL

To begin with, we shall confine our attention to nets having a single output line and arbitrary number of input lines. Each line will be capable of assuming only one of the two possible values at a time, namely 0 or 1. Each net will consist of two parts, as shown in Figure 10.

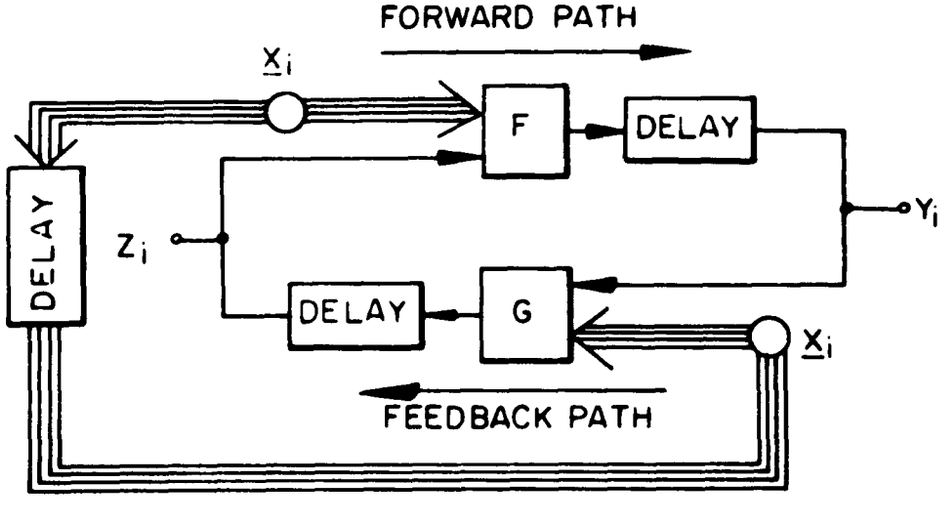


Fig. 10 The Block Diagram of a Self-Correcting Net.

The part F is in the forward path and the part G is in the feedback path. The delay units represent the intrinsic delays involved in the calculations of F and G, and in addition, may also have additional periods of delay introduced for the purpose of synchronization with a master controller, or for whatever other reasons that might exist. The feedback net G is the error-correcting net. For each input valuation $\underline{X}_i$ of the net F, the net G will decide whether the input-output pair, $(\underline{X}_i, y_i)$, satisfies a desired logical function f_D and generate an appropriate control signal, x_i, that would condition F suitably for the next calculation. Of course, the net G is also likely to generate erroneous outputs. However, when z_i is correct its value will indicate whether the pair $(\underline{X}_i, y_i)$ is a proper pair or not.

For a given fixed input, $\underline{X}_i$, the net in Figure 10 will produce a computation (a sequence of calculations) consisting of pairs of outputs (y_i, z_i). In this computed sequence, the value of z_i in each pair will indicate whether the corresponding y_i is the correct output or not. In general, instead of keeping the input, $\underline{X}_i$, fixed it could be encoded into an input sequence and in this case a sub-sequence of the computer output sequence would be the required, correct output sequence. In the following discussion we shall assume that the input $\underline{X}_i$ is in the uncoded form — i.e., is held fixed for a given computation. It is our opinion that computation with encoded inputs would not cause any additional advantage in reliability of operation.

Let the mappings performed by the F and G nets be denoted by the logical functions f_F^{m+1} and f_G^{m+1}, where m of the inputs are external inputs, the (m+1)st being the feedback input:

$$f_F^{m+1}(\underline{X}_i, z_i) = y_i \tag{3}$$

$$f_G^{m+1}(\underline{X}_i, y_i) = z_i \tag{4}$$

The functions F and G may be expanded as follows*:

$$f_F^{m+1}(\underline{X}_i, z_i) = g_{11}^m(\underline{X}_i) \cdot z_i + g_{12}^m(\underline{X}_i) \cdot \bar{z}_i \tag{5}$$

$$f_G^{m+1}(\underline{X}_i, y_i) = g_{21}^m(\underline{X}_i) \cdot y_i + g_{22}^m(\underline{X}_i) \cdot \bar{y}_i \tag{6}$$

* '.' denotes logical AND, '+', logical OR and '-' denotes complementation.

where the g's are functions of m arguments. To specify completely the computation performed by such a finite sequential machine it is necessary to fix a distinguished initial state (namely, the contents of the delay units at time 0) and an end condition for stopping the calculations. Also, the sequence of output pairs, (y,z), should be properly indicated as functions of time.

Assuming that the delay introduced by the delay units in Figure 10 is of unit magnitude and the net as a whole is periodically activated to perform a calculation at the beginning of every unit interval of time, the equations (5) and (6) may be written as follows:

$$y_n = a_{1n}(\underline{X}_i) \cdot z_{n-1} + a_{2n}(\underline{X}_i) \cdot \bar{z}_{n-1} \tag{7}$$

and

$$z_n = b_{1n}(\underline{X}_i) \cdot y_{n-1} + b_{2n}(\underline{X}_i) \cdot \bar{y}_{n-1} \tag{8}$$

where $n = 1, \ldots$ is the time parameter, z_n, y_n are the n^{th} values of the outputs, and a_{1n}, a_{2n}, b_{1n} and b_{2n} represent the 'g' functions of equations (5) and (6) at the instant n. In general, it is thus possible that as n varies the g functions may also vary. Let us now see how the initial and final conditions may be specified for the equations (7) and (8) and how the resultant computation may be used to calculate a <u>desired</u> logical function f_D^m.

The end conditions may be specified in one of two possible ways. One can fix the length of computation for a given input to be the same number, N, and specify a rule for selecting the appropriate output from the N calculations. This, of course, will require additional equipment to perform the selection; and, further, in this scheme one does not take advantage of the probabilities of distribution of occurrence of errors to increase the speed of calculation. In fact, the selection procedure will be consuming additional time for completing the calculation.

On the other hand, one may decide to terminate the calculations for a given input, on some criterion depending upon the relative values of the outputs (y,z). There are only three distinct possibilities in which this

may be done, and these are*: (Note that the corresponding outputs, as shown in Figure 9, are always separated by one unit of delay.)

$$(1)\quad y_n = z_{n+1} \rightarrow y_n = f_D(\underline{X})$$

$$(2)\quad z_{n+1} = 1 \rightarrow y_n = f_D(\underline{X})$$

$$(3)\quad z_{n+1} = 0 \rightarrow y_n = f_D(\underline{X}).$$

Permutation of y and z in the above three conditions does not change the essential nature of the constraints.

In case (1) an external comparator is necessary to determine that y_n and z_{n+1} are of the same value and generate the appropriate end signal. The remaining cases presume the existence of some form of verification within the net; and, hence, the proper value of z_{n+1} may itself be used as the terminating signal. In all the three cases an erroneous output $y_n \neq f_D(\underline{X})$ will be chosen only if both y and z are simultaneously in error. Also, for a given input, calculations will be continued only as long as there is an error in the output; and this feature, if properly used, will result in the saving of time.

In the following discussions we shall consider only cases (2) and (3) for the end condition, as these seem to take into account more comprehensively than the rest all the equipment needed for the calculation. It should, of course, be noted that in our discussions so far, we have implicitly assumed that there is an external master controller that provides, periodically, the clocking pulses needed for starting each calculation.

Also, as the length of every computation is not the same, it is necessary to introduce a facility to gate out the proper result at the termination of a computation and to gate in a new set of inputs for the next computation. In cases (2) and (3) of the end condition, the terminating value, z_n, may be used to start the gating operations at the input and output ends of the unit. In such an arrangement the terminating value, z_n, may also be the initial value, z_o, for the next computation. Let the initial value of y, namely y_o, be chosen to be the same as the value of the previous output. Therefore, y_o may be either 1 or 0. This arrangement eliminates the necessity

---

* The reader may easily verify this assertion.

for resetting the initial values at the beginning of each computation. The problem now is to get the conditions on the functions a_{1n}, a_{2n}, b_{1n} and b_{2n} in order to calculate a desired function, f_D, with the above model of a sequential net, using the smallest necessary number of calculations. We shall see that the imposition of the restriction of minimum computational delay forces the functions a_{1n}, a_{2n}, b_{1n}, and b_{2n} to assume the value of either $f_D^m(\underline{X})$ or $\bar{f}_D^m(\underline{X})$, for all n.

C. DETERMINATION OF THE FUNCTIONS a_{1n}, a_{2n}, b_{1n}, b_{2n}

Before proceeding with the analysis let us take stock of the assumptions we have made so far, and the limitations these assumptions will impose on the general validity of whatever conclusions we may arrive at.

We have now three requirements on the computation performed by the net of Figure 10. These are

(i) The net should start each computation from one of the prescribed initial states. (Notice, however, that each calculation of the net might start from any one of the four possible initial states).

(ii) A computation is terminated when the end condition is satisfied. At the termination of a computation the selected output value should be equal to $f_D(\underline{X})$.

(iii) In order to minimize computational delay the net should seek to satisfy the end conditions in each calculation. Also, after termination, the selected output of the computation must be equal to $f_D^m(\underline{X})$.

The initial state of a computation was chosen to be the same as the terminating state of the previous computation. This was possible because the terminating state was used only to do the proper gating at the input and output ends, and not to terminate the operation of the net itself. The other possibility, namely resetting the initial state to a fixed distinguished value at the beginning of each computation will require additional equipment to do the resetting operations. This will only increase the complexity of the net, thereby reducing its reliability of operation. Therefore, the choice we made is preferable.

Out of the three possible end conditions, the latter two take into account, more comprehensively than the first, all the operations that the net should perform to complete a computation. Therefore we shall confine our attention only to the end conditions (2) and (3). In fact, if an external comparing device is introduced in the net G of Figure 10, for the end condition (1), to generate the terminating signal, then the end condition (1) will reduce to one of the remaining two conditions. Thus, analysis of the net for conditions (2) and (3) should exhaust all the possible terminations with which the net might be operated.

The third requirement is necessary to satify the condition that each computation should be of the shortest possible length. If this requirement is not imposed, it is not clear a priori, what function the net should calculate in each operation.*

The operation of the net satisfying the above three requirements, for the end conditions (2) and (3) is described by tables I and II, respectively. These tables may be read as follows:

Table I:

(i) Design Requirement: Each computation should be of the shortest possible length.

(ii) End Condition: $z_{n+1} = 1 - y_n = f_D(\underline{X}_i)$.

(iii) Initial Condition: $y_o = 0$ or 1 and $z_o = 1$.

y_n	z_n	$f_D(\underline{X}_i)$	y_{n+1}	z_{n+1}	$a_{1n}(\underline{X}_i)$	$a_{2n}(\underline{X}_i)$	$b_{1n}(\underline{X}_i)$	$b_{2n}(\underline{X}_i)$
0	1	1	1	0	1	*	*	0
0	1	0	0	1	0	*	*	1
1	1	1	1	1	1	*	1	*
1	1	0	0	0	0	*	0	*
0	0	1	1	0	*	1	*	0
0	0	0	0	1	*	0	*	1
1	0	1	1	1	*	1	1	*
1	0	0	0	0	*	0	0	*

* It is likely that the function to be calculated by a net, in each calculation, may be determined through some conditions, connected with error diagnosis. In such a case the requirement that each computation be of the minimum possible length, will not be satisfied.

Table II:

(i) Design Requirement: Same as Table I

(ii) End Condition: $z_{n+1} = 0 \rightarrow y_n = f_D(\underline{X}_i)$.

(iii) Initial Condition: $y_o = 0$ or 1 and $z_o = 0$.

y_n	z_n	$f_D(\underline{X}_i)$	y_{n+1}	z_{n+1}	$a_{1n}(\underline{X}_i)$	$a_{2n}(\underline{X}_i)$	$b_{1n}(\underline{X}_i)$	$b_{2n}(\underline{X}_i)$
0	0	1	1	1	*	1	*	1
0	0	0	0	0	*	0	*	0
1	0	1	1	0	*	1	0	*
1	0	0	0	1	*	0	1	*
0	1	1	1	1	1	*	*	1
0	1	0	0	0	0	*	*	0
1	1	1	1	0	1	*	0	*
1	1	0	0	1	0	*	1	*

Let us first consider the Table I. The first two columns of this table give the values of the outputs at the instant n, for n = 0, 1, ... The third column gives the value of $f_D(\underline{X}_i)$, for the input $\underline{X}_i$ at the instant n. The fourth and fifth columns give the expected values of the outputs at the next instant, to satisfy the requirements that the proper computation should be of the shortest possible length. For these input and output conditions one may calculate from equations (7) and (8) the values that the functions 'a' and 'b' must assume, for the input $\underline{X}_i$.

Consider, for example, the first row of Table I. For the initial conditions shown, one gets from equations (7) and (8) that for n = T,

$$y_{T+1} = a_{1T}(\underline{X}) = 1 \tag{9}$$

$$z_{T+1} = b_{2T}(\underline{X}) = 0 \tag{10}$$

and $a_{2T}(\underline{X})$ and $b_{1T}(\underline{X})$ might assume any values whatsoever. As $z_{T+1} = 0$ the end condition is not satisfied, and, therefore, the net goes through another

calculation for the same input $\underline{X}_i$; only it starts from the initial state, $(y,z) = (1,0)$, instead of $(0,1)$. The operation of the net, this time, is described by the last but one row of Table I. The final outputs are, [from equations (7) and (8)]

$$y_{T+2} = a_{2(T+1)}(\underline{X}_i) = 1 \tag{11}$$

$$z_{T+2} = b_{1(T+1)}(\underline{X}_i) = 1, \tag{12}$$

and these do not depend on the values assumed by $a_{1(T+1)}(\underline{X}_i)$ and $b_{2(T+1)}(\underline{X}_i)$, as indicated by the '*' in the table. The output pair (y_{T+1}, z_{T+2}) now satisfies the end condition for Table I and as such the computation is terminated. For other initial states the table is similarly filled in. Table II is constructed similarly, with the only difference that the end condition is now different. The following observation concerning the operations shown in Tables I and II are now pertinent.

1. Comments on Tables I and II

Comment 1 The value, $z_n=1$, in Tables I and II fixes the value of $a_{1n}(\underline{X}_i) \equiv f_D(\underline{X}_i)$ for all $\underline{X}_i$. When $z_n = 0$, $a_{1n}(\underline{X}_i)$ would still have the same value, unless, of course, the net corresponding to a_{1n} was itself changed by the change in the value of z_n. However, when $z_n = 0$, the output is independent of $a_{1n}(\underline{X}_i)$, as may be verified from equation (7). So one would find no reason to modify the net of a_{1n} as a function of z_n. Thus, the net may be kept fixed for all time, n; no additional advantage will be gained by making the net adjustable.

Similarly $z_n = 0$ fixes the function $a_{2n} \equiv f_D$ for all $\underline{X}_i$ and for all time n, and in this case ($z_n = 0$) the output is independent of $a_{1n}(\underline{X}_i)$. Therefore, if two separate, identical nets were used, one for a_{1n} and another for a_{2n}, then depending upon the value of z_n the output, y_{n+1}, will depend on one or the other of the two nets. The input z_n may be used to switch the operation from one net to the other depending upon which one produced a wrong output. This feature introduces the self-correcting property in nets of this type.

Likewise, $y_n = 0$, fixes $b_{2n} \equiv \bar{f}_D$ in Table I and $b_{2n} = f_D$ in Table II; $y_n = 1$, sets $b_{1n} \equiv f_D$ in Table I and $b_{1n} \equiv \bar{f}_D$ in Table II.

For these values of 'a' and 'b' functions equations 7 and 8 may be written, for the end condition (2) as

$$y_{n+1} = f_D(\underline{X}) \cdot z_n + f_D(\underline{X}) \cdot \bar{z}_n \tag{13}$$

$$z_{n+1} = f_D(\underline{X}) \cdot y_n + \bar{f}_D(X) \cdot \bar{y}_n \tag{14}$$

and for the end condition (3),

$$y_{n+1} = f_D(\underline{X}) \cdot z_n + f_D(\underline{X}) \cdot \bar{z}_n \tag{15}$$

and $$z_{n+1} = \bar{f}_D(\underline{X}) \cdot y_n + f_D(\underline{X}) \cdot \bar{y}_n \tag{16}$$

The use of the same function f_D more than once in the above equations signifies the fact that each value would be calculated by a distinct logical net.

The above values of the functions, to be calculated by each net in each calculation, were obtained because of the restrictions imposed on the required values of y_{n+1} and z_{n+1} in columns 4 and 5 of Tables I and II. The values of y_{n+1} and z_{n+1} were chosen for each calculation such that the whole computation will be terminated with the shortest possible delay, producing the correct output $y_n = f_D^m(\underline{X})$. One may easily verify that the above assignment is the only possible one, that would minimize the length of each computation.

Comment 2. For no two distinct initial states, (y_n, z_n) the same pair of functions (a_{in}, b_{jn}), $(i,j = 1 \text{ or } 2)$ is used in the calculations. Therefore, unless an erroneous operation produces at the instant $(n + 1)$ the same erroneous outputs as at the instant n, a new calculation is always done using a new net.

Comment 3. The equations (14) and (16) represent just modulo 2 comparison functions, which may be written as

$$z_{n+1} = \bar{f}_D(\underline{X}) \oplus y_n \qquad (14)a$$

and

$$z_{n+1} = f_D(\underline{X}) \oplus y_n, \qquad (16)a$$

where $\oplus$ indicates the 'Exclusive OR' operation. The feedback network therefore, compares the value of y_n with the required value, $f_D(\underline{X})$, and gives a command for recalculation whenever $y_n = f_D(\underline{X})$.

In fact, in the discussion in section R., on the model shown in Figure 6 we had required of the net some such similar operation. There we said that the feedback input would condition the net suitably for the next calculation. The above arguments indicate that the only possible useful conditioning that may be done will be to give a command for recalculation, possibly using an alternate net. This conclusion will be valid, of course, only if the requirement of minimum computational delay is imposed on the net. At present, it is not clear, what other general requirement may be substituted in its place. The initial and final conditions were specified, in a sense, in a very natural way. In this case, it is clear that there are no other boundary conditions that could be more profitably used.

We believe that even in the case of computation with encoded inputs the same conclusion would be valid. However, it is difficult to prove this conjecture in this generality, because it is not clear what general restrictions exist on codes for computation. (In an analogous situation in the case of communication systems, a well-known set of general restrictions on possible efficient codes for error-correction is provided by the properties of group codes).

2. Generalization of the Concepts:

The above arguments can be directly extended to the case where there is more than one output line for each net F and G of Figure 10. Let the output lines of F be denoted by

$$\underline{Y} = (y_1\ y_2 \cdots y_t)\ ,\ t \geq 1$$

and those of G by

$$\underline{z} = \{z_1\ z_2\ \cdots\ z_t\}\ ,\ t \geq 1.$$

Let $\underline{y}_i$ and $\underline{z}_i$ denote particular valuations of $\underline{y}$ and $\underline{z}$, where

$$\underline{y}_i = \{y_{1i}\ y_{2i}\ \cdots\ y_{ti}\}$$

and

$$\underline{z}_i = \{z_{1i}\ z_{2i}\ \cdots\ z_{ti}\}.$$

$\underline{y}_i$ and $\underline{z}_j$ for $i,j = 0,1, \ldots 2^t-1$ will denote the 2^t valuations that $\underline{y}$ and $\underline{z}$ might assume. Let Y_i and Z_j be the values of a logical function, defined on $\underline{y}$ and $\underline{z}$ as follows:

$$Y_i = 1 \rightarrow \underline{y} = \underline{y}_i$$

and

$$Z_i = 1 \rightarrow \underline{z} = \underline{z}_i$$

For each pair of output lines y_k and z_k, $k = 1, 2, \ldots, t$ their values $y_k\ (n+1)$ and $z_k\ (n+1)$ at the instant $(n+1)$ will be given by an equation of the form*

$$y_k\ (n+1) = \sum_{\ell=0}^{2^t-1} a_{k\ell n}\ (\underline{X}).Z_\ell(n),\ k = 1, \ldots t \qquad (17)$$

and

$$z_k\ (n+1) = \sum_{\ell=0}^{2^t-1} b_{k\ell n}\ (\underline{X}).Y_\ell(n),\ k = 1, \ldots t. \qquad (18)$$

In the above equations $a_{k\ell n}(\underline{X})$ and $b_{k\ell n}(\underline{X})$ are the functions that generate the k^{th} outputs, y_k and z_k, respectively, at the instant $n = 0, 1, \ldots$. For $\ell = 0, 1$, and $k = 1$ the above equations will reduce to the form of equations (7) and (8). (In (7) and (8) the subscript k was not necessary).

---

* $\sum$ denotes a logical 'OR' summation.

At any instant n, if the output valuations were $\underline{y}_i(n)$ and $\underline{z}_j(n)$. $0 \leq i, j \leq 2^t-1$ then from equations (17) and (18) we get:

$$y_k \ (n+1) = a_{kjn} \ (\underline{X}); k = 1, \ \ldots \ t, \tag{19}$$

$$z_k \ (n+1) = b_{kjn} \ (\underline{X}); k = 1, \ \ldots \ t, \tag{20}$$

where $\underline{X}$ would be the input at the instant n. Thus for <u>each</u> pair of output vectors $(\underline{y}_i, \underline{z}_j)$ the <u>two</u> nets corresponding to the functions a_{kjn} and b_{kin} are chosen for calculation, for <u>each</u> k = 1, ...t. For the same reasons discussed in section C.1, comment 1, the functions 'a' and 'b' may be chosen to be independent of the time parameter, n. Also, for each k, $1 \leq k \leq t$ if the desired output function is $f_{D_k}(\underline{X})$. Then from requirement 3* we will obtain that

$$a_{kj} = f_{D_k}$$

for all j = 0, 1, ..., 2^t-1, and all n = 1, 2, ...

For any given value, $\underline{y}_i(n)$, the value $\underline{z}_j(n+1)$, may be used to indicate whether $\underline{y}_i(n)$ is correct, and if it is not correct, which one of the 2^t-1 possible errors is present in $\underline{y}_i(n)$. In this case, therefore, the design of the nets corresponding to the functions $b_{k\ell}$ will depend on the nature of error diagnosis that is desired. It should be valuable if some general conditions could be obtained on the specification of the functions $b_{k\ell}$.

It is not clear at present, what the error-correcting potential of this scheme will be, and taking into consideration all the extraneous control nets, what the overall gain in reliability will be.

The values of the probability of occurrence of a proper computation and that of an erroneous computation for some typical nets are calculated in the next section. Also, the expected length of a proper computation, as well as an improper one, are obtained. We shall discuss the results and their significance, in detail, in the next section. The following general comments may be made concerning the scheme, at this moment:

* Minimum computational delay

It is clear that the process of substitution of a new net in place of another that generated an erroneous output, indicates that, with this scheme it should be possible to combat both type 1 (temporary) and type 2 (permanent) errors.

Also, there is the possibility of developing error-diagnostic nets which will prove to be of great value in any large data processing system.

It appears that if more redundant alternate nets were available in a net, then greater reliability of operation would be realizable. Also, the principal concept of substituting an alternate net in place of an erroneous one, is very elementary in error-correction. The above scheme seems to offer a natural way of doing this, in practice.

We believe that, the model discussed so far, provides a good context in which further investigation may be made to understand the process of self-correction and possibly develop general theoretical concepts on error-diagnosis and self-repair. We shall be interested in answering questions of the following nature:

1. For a given number of feedback lines what boundary conditions will yield the maximum reliability? The selection of an appropriate boundary condition would be influenced by the following considerations:

The number of additional operations necessary to terminate one computation and start the next computation.

The nature of error-signalling, if any, that is expected of the net.

The cost of termination in terms of loss in reliability.

2. If the precise nature of possible errors were known, is there any significance in changing the logical function for different calculations in a computation?

In the above discussion we assumed the logical function for each calculation to be $f_D(\underline{X})$. By changing the logical function for a given net, it may be possible to perform error-diagnosis.

3. In a large net composed of subnets of the form discussed so far, the total net is essentially asynchronous, because, depending upon the occurrence of errors, each subnet may take a different time to complete its computation. This calls for a proper scheduling of each operation or provision for buffer storages at the input of each subnet. Depending upon the probabilities of occurrence of errors, a queuing problem arises. The length of the queue at the input of each subnet will directly be a measure of the performance of the net. It remains to be seen how this property can be exploited to do error-prediction in a systematic way. In the design of a large net one's aim should be to minimize the length of the queue at the input of each subnet.

4. In every net of this form a possibility exists that the control net might fail, causing a computation to be continued indefinitely without termination. To combat this type of failure one may assign a maximum length N for each computation. After N calculations, it will, therefore, be necessary to adopt some type of statistical estimate to choose the correct output. A study of this will also be a part of a study of self-correcting system.

These problems arise in a natural way in the context of the self-correcting scheme discussed in this report. It is our opinion that a study of these problems in greater depth would be a worthwhile task.

D. THE ANALYSIS OF DYNAMIC BEHAVIOR OF A SELF-CORRECTING NET

The dynamic behavior of a self-correcting net may be described by the probabilities of occurrence of the following:

(1) A proper computation of length ℓ.

(2) An erroneous computation of length ℓ.

(3) A computation being of length greater than ℓ.

Let the respective probabilities of events (1), (2), and (3) be denoted by $P_p(\ell)$, (p stands for 'proper'), $P_e(\ell)$, (e for erroneous) and $P_c(\ell)$, (c for continuation). The overall performance of the net may be evaluated from:

$$P_p = \sum_{\ell=1}^{\infty} P_p(\ell) \tag{21}$$

$$P_e = \sum_{\ell=1}^{\infty} P_e(\ell) \tag{22}$$

where P_p is the total probability of occurrence of a proper computation of any length, and so also, P_e, that of an erroneous computation. It is clear that,

$$P_c(\ell - 1) = P_p(\ell) + P_e(\ell) + P_c(\ell), \tag{23}$$

and, therefore, from equations (21), (22) and (23),

$$\sum_{\ell=1}^{\infty} P_c(\ell - 1) - \sum_{\ell=1}^{\infty} P_c(\ell) = P_c(0) = P_e + P_p = 1 \ . \tag{24}$$

Also, it is true that

$$\lim_{\ell \to \infty} P_c(\ell) = 0 \tag{25}$$

The expected values and variances of the lengths of a proper, as well as an erroneous computation are of significance in evaluating the performance of such nets. Let $E(\ell_p)$ and $E(\ell_e)$ denote the respective expected values. So also, let $V(\ell_p)$ and $V(\ell_e)$ denote their variances:

$$E(\ell_p) = \{\sum_{\ell=1}^{\infty} \ell \cdot P_p(\ell)\} / P_p \tag{26}$$

$$E(\ell_e) = \{\sum_{\ell=1}^{\infty} \ell \cdot P_e(\ell)\} / P_e \tag{27}$$

$$V(\ell_p) = \{\sum_{\ell=1}^{\infty} [\ell - E(\ell_p)]^2 \, P_p(\ell)\} / P_p \tag{28}$$

$$V(\ell_e) = \{\sum_{\ell=1}^{\infty} [\ell - E(\ell_e)]^2 \, P_e(\ell)\} / P_e. \tag{29}$$

For given component reliabilities a good design for a self-correcting net should minimize P_e, $E(\ell_p)$ and $V(\ell_p)$. Also it will be desirable to have

$$E(\ell_e) > E(\ell_p). \tag{30}$$

In fact, the most desirable situation will be:

$$E(\ell_e) - \sqrt{V(\ell_e)} > E(\ell_p) + \sqrt{V(\ell_p)} \tag{31}$$

The above conditions will assure us that a proper termination of a computation is always more likely to occur than an erroneous one.

We shall now discuss the analytical techniques that might be employed to calculate the above statistical parameters of a self-correcting sequential net.

1. Temporary Errors: (Type I)

(a) The Mathematical Model:

In the case of temporary errors the operation of a self-correcting net may be described by a Markov chain diagram, as shown in Figure 11.

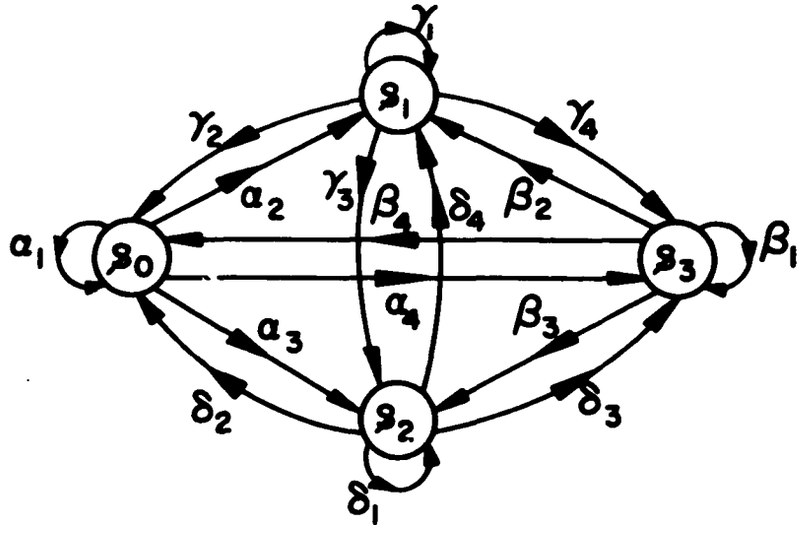


Fig. 11 A Markov chain diagram of a self-correcting net.

Each node in this figure represents a possible output state of the net at an instant, n. A directed branch, between a pair of nodes, indicates a possible transition, in the direction of the branch, from one output state at an instant n, to the other at the next instant, n + 1*. A self-loop indicates that it is possible to have the same output state at successive instants of time. The weight on each branch indicates the probability of occurrence of the transition represented by the branch. The probabilities ξ_i for $\xi = \alpha, \beta, \gamma, \delta$ and $i = 1, 2, 3, 4$, in Figure 11, will be functions of the input of the net at an instant n, and

$$\sum_{i=1}^{4} \xi_i = 1 \qquad \text{for} \quad \xi = \alpha, \beta, \gamma, \text{ and } \delta \tag{32}$$

Such a net will have a subset of states, say S_o, called the set of distinguished initial states. For any input, $\underline{X}$, a computation by the net will always start from one of the distinguished initial states, $s_{oi} \in S_o$. Depending upon the terminating conditions, one or more of the output states of the diagram will be chosen as the terminating states of the net. A computation by the net will always terminate in one of the possible terminating sates. Let the set of all possible terminating states be denoted by S_T. A proper computation may be defined in one of the following ways:

Definition 1. (Proper computation):

(1) A computation, whose terminating state $s_{pi} \in S_p(\underline{y}) \subset S_T$, where $S_p(\underline{y})$ is the subset of proper terminating states, for a desired output $\underline{y}$.

(2) A computation having one or more occurrences of certain specially chosen output states, at certain well-defined places in the computed output sequence, prior to termination. Let these distinguished intermediate states be denoted by $S_I(\underline{y})$, for a desired output $\underline{y}$.

(3) A computation satisfying both conditions (1) and (2).

---

* It is assumed that it takes one unit of time to complete a transition.

Similarly, a subset of terminating states corresponding to an Erroneous computation may be denoted by $S_E(\underline{y}) \subset S_T$. Also, a computation may be defined to be erroneous if no state $s_i \in S_I(\underline{y})$ occurs as an intermediate state, at the proper places, in the course of a computation. The sets $S_p(\underline{y})$, $S_E(\underline{y})$, $S_I(\underline{y})$ and S_T will satisfy the following set relationships:

$$S_p(\underline{y}) \cup S_E(\underline{y}) = S_T \tag{33}$$

$$S_p(\underline{y}) \cap S_E(\underline{y}) = \phi^* \tag{34}$$

and

$$S_I(\underline{y}) \cap S_T = \phi \ . \tag{35}$$

The set, S_o, of distinguished initial states may be chosen arbitrarily.

For a given input, $\underline{X}_i$, if $f_D(\underline{X}_i) = \underline{Y}_i$ (f_D is the desired function), then the probabilities of individual transitions in the Markov chain diagram of the net will change as a function of $\underline{X}_i$, such that the probability of occurrence of a proper computation would be maximized. As the number of states in the diagram increases the number of possible paths for proper, as well as an erroneous computation will increase rapidly. The variety of ways in which the different subsets of equations (33), (34) and (35) may be specified will be very large too. In such cases, it will be desirable to find some general conditions on the selection of the subsets that will maximize, P_p and satisfy the inequality (31). At present, we do not have a good understanding of this problem. However, it is clear that a theory of self-correcting systems must be capable of answering such questions.

(b) Techniques for Analyzing Model (A)

Convenient techniques for obtaining the probability of occurrence of a specified event in a stationary Markov chain have been developed by K. Kaplan[18] and J. Sklansky of RCA Laboratories. In this note we shall briefly explain their procedures in order that the reader may understand easily the calculations that are to follow :

A Markov chain diagram may be interpreted as a signal flow graph, where each branch weight will denote the signal gain provided to the signal that is transmitted through the path represented by the branch.

* In case the condition 1 of Definition 1 is not used this property need not be satisfied.

At each node, the signals coming into the node through the various paths of the graph are added up and retransmitted through the outgoing paths of the node. The sum of the incoming signals is always equal to the sum of the outgoing signals.

In such a graph, if a unit impulse is injected into a node, say N_o, from an external source, at time 0, one may calculate, using the z-transform techniques, the magnitude of the signal that will be present at any given node, say N_i, at a given instant, say n, because of signal transmissions through the various paths of the graph. This magnitude of the signal at N_i, at the instant n, will be equal to the probability of occurrence of the following event, in the corresponding Markov chain:

The event that, in an experiment on the Markov chain, starting at time 0, from a state, say s_o, represented by the node N_o in the graph, the state s_i, represented by N_i, occurs at the instant n.

For a proof of this property and its consequences the reader is referred to references 18 and 19. Using the above property one may obtain expressions in closed form, in terms of the individual transition probabilities, for the various probabilities and expected values values introduced in section D.

2. Permanent Errors (Type II)

In the case of a permanent failure of a subnet, the ability of a self-correcting net to successfully repair the failure will depend upon the number of alternate subnets that have been provided to the net, a priori. In this case, the calculation of the probability of occurrence of a permanent failure in the total net is straightforward and no special mathematical techniques are called for.

In the next section we shall introduce three self-correcting nets and calculate the various parameters associated with them. These examples are intended primarily to illustrate the problems involved in the construction of such nets. Also, an understanding may be obtained of the orders of magnitude of the different quantities and how they may be changed with changes in individual output transition probabilities.

3. Description of Some Typical Self-Correcting Nets

The outputs y_n and z_n of the net, $\mathfrak{F}_1$, in Figure 12 are given by the equations:

$$y_n = f_D(\underline{X}) \cdot z_{n-1} + f_D(\underline{X}) \cdot \bar{z}_{n-1} \tag{13}$$

$$z_n = f_D(\underline{X}) \cdot y_{n-1} + \bar{f}_D(\underline{X}) \cdot \bar{y}_{n-1} \cdot \tag{14}$$

The boxes F_1 and G_1 of the net $\mathfrak{F}_1$ correspond to the F and G boxes of Figure 10. Inside F_1 and G_1 the nets D_1, D_2, D_3 and D_4 denote the D-nets that calculate the desired function, $f_D(\underline{X})$. These four nets correspond to the four occurrences of the function $f_D(\underline{X})$ in equations (13) and (14). Indicating this correspondence more explicitly, equations (13) and (14) may be rewritten as:

$$y_n = f_{D_1}(\underline{X}) \cdot z_{n-1} + f_{D_2}(\underline{X}) \cdot \bar{z}_{n-1} \tag{36}$$

$$z_n = f_{D_3}(\underline{X}) \cdot y_{n-1} + \bar{f}_{D_4}(\underline{X}) \cdot \bar{y}_{n-1} \tag{37}$$

The gate, marked C in Figure 12 is a complementing gate. The markings on the other gates are self-explanatory. In this net, a typical output state will be a valuation of the pair (y,z). The net has, in all, only four states, viz: (0,0), (0,1), (1,0) and (1,1). The sets S_o, $S_T(y)$ and $S_I(y)$ for $y = 0, 1$, are:

$$\begin{aligned} S_o &= \{(0,1), (1,1)\}. \\ S_T(0) &= S_T(1) = S_o \quad * \\ S_I(0) &= (0,0) \\ \text{and} \quad S_I(1) &= (1,0). \end{aligned} \tag{38}$$

In this case the terminating condition has been chosen as

$$z_{n+1} = 1 \rightarrow y_n = f_D(\underline{X})$$

and, therefore, the terminating state itself is not used to distinguish between a proper and an erroneous computation. When $y = 0$, a computation will be

* In this case the condition 1 of Definition 1 has not been used.

proper only if the state $S_I(0) = (0,0)$ occurs as the last but one state of the computation. So also, when $y = 1$ the state $S_I(1) = (1,0)$ should occur as the last but one state, for a proper computation. An erroneous computation will occur only if $\{z_{n+1} = 1$ and $y_n \neq f_D(\underline{X})\}$ for some $n = 1, 2, \ldots$. The probability of occurrence of this event will give us the first measure of reliability of the net, $\mathcal{F}_1$.

Let*

$$p'(D_j) = \alpha, \quad j = 1, 2, 3, 4 \tag{39}$$

be the first measure of unreliability for the nets D_j, $j = 1, 2, 3, 4$ in Figure 12. So also, let

$$p'(\&) = p'(OR) = p'(C) = \beta \tag{40}$$

be the first measure of unreliability of the other gates in Figure12, that belong to the control part of the self-correcting net. One can show that the first measure of unreliability of the net F_1 will be:

$$p'(F_1) < (\alpha + 3\beta) = \epsilon_1 \text{ say.} \tag{41}$$

For the purpose of our calculations we shall choose the upper bound ϵ_1 as equal to $p'(F_1)$.

The net G_1 may cause two types of errors, type A and type B:

Type A: is the event **

$$\{z_{n+1} = 0 | y_n = f_D(\underline{X})\} .$$

This causes a recalculation even though $y_n = f_D(\underline{X})$, thereby increasing the length of a computation. This type of error does not directly cause an erroneous computation. Let us denote the probability of this event by ϵ_A.

---

* $p'(D_j) = \underset{i}{\text{Max}}\ p_i$ where p_i is defined by:

$$f_{D_j}(\underline{X}_i) \quad \begin{cases} = \bar{f}_D(\underline{X}_i) \text{ with probability } p_i \\ = f_D(\underline{X}_i) \text{ with probability } (1-p_i) \end{cases}$$

for the input, $\underline{X}_i$.

** To be read as $z_{n+1} = 0$ given that $y_n = f_D(\underline{X})$.

Type B: is the event

$$\{z_{n+1} = 1 \mid y_n \neq f_D(\underline{X})\}.$$

This error causes the computation to be erroneous. Let, ϵ_B, be the probability of this event.

For the net G_1 the probabilities of occurrence of these two events can be shown to be:

$$p\{z_{n+1} = 0 \mid y_n = f_D(\underline{X})\} = P\{z_{n+1} = 1 \mid y_n \neq f_D(\underline{X})\} < \epsilon_1 \qquad (42)$$

We shall choose in our ensuing calculations

$$p'(F_1) = p'(G_1) = \epsilon_1 \qquad (43)$$

for the net $\mathcal{F}_1$, of Figure 12.

The net $\mathcal{F}_2$ in Figure 13 is a modified form of the net in Figure 12. In $\mathcal{F}_2$, greater redundancy has been introduced in the calculation of a_{n+1}:

$$z_{n+1} = z^1_{n+1} \cdot z^2_{n+1} \qquad (44)$$

where

$$z^1_{n+1} = f_{D_3}(\underline{X}) \cdot y_n + \bar{f}_{D_4}(\underline{X}) \cdot \bar{y}_n \qquad (45)$$

$$z^2_{n+1} = f_{D_4}(\underline{X}) \cdot y_n + \bar{f}_{D_3}(\underline{X}) \cdot \bar{y}_n \qquad (46)$$

In this scheme, for a given value of y_n both D_3 and D_4 will be used to calculate z_{n+1}. If one of the quantities z^i_{n+1}, $i = 1, 2$, indicate that y_n is erroneous (i.e., $z^i_{n+1} = 0$ for $i = 1$ or 2) the computation will be continued. A termination will occur if and only if both z^1_{n+1} and $z^2_{n+1} = 1$. For this net,

$$p'(F_2) = \epsilon_1 \qquad (47)$$

and

$$P\{z_{n+1} = 0 \mid y_n = f_D(\underline{X})\} < \epsilon_1(1 - 2\beta)(2 - \epsilon_1) + \beta = \epsilon_A \text{ say,} \qquad (48)$$

$$P\{z_{n+1} = 1 \mid y_n = f_D(\underline{X})\} < \epsilon_1^2(1 - 2\beta) + \beta = \epsilon_B \text{ say,} \qquad (49)$$

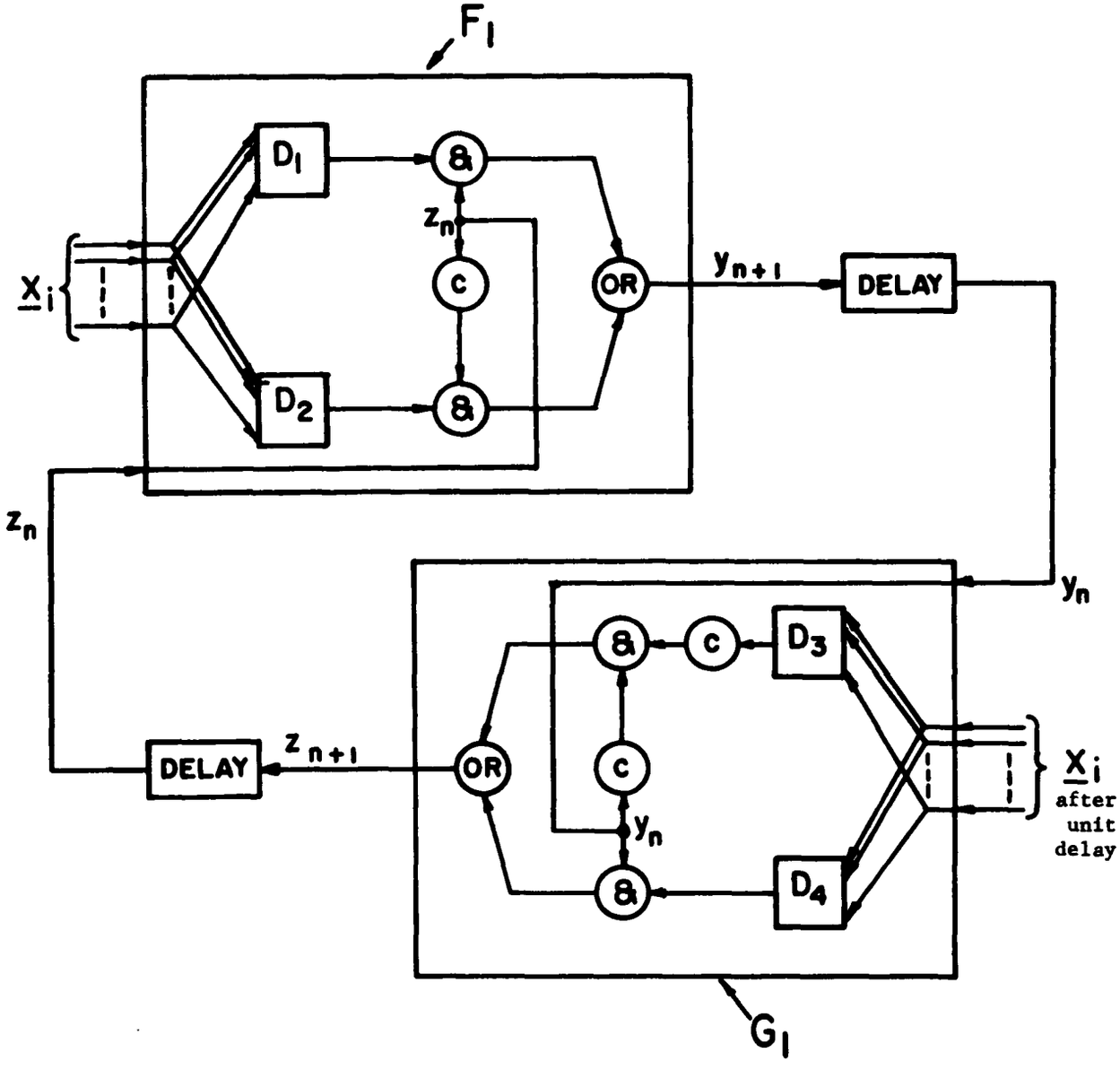


Fig. 12. The net, $\mathcal{F}_1$, described by equations (13) and (14)

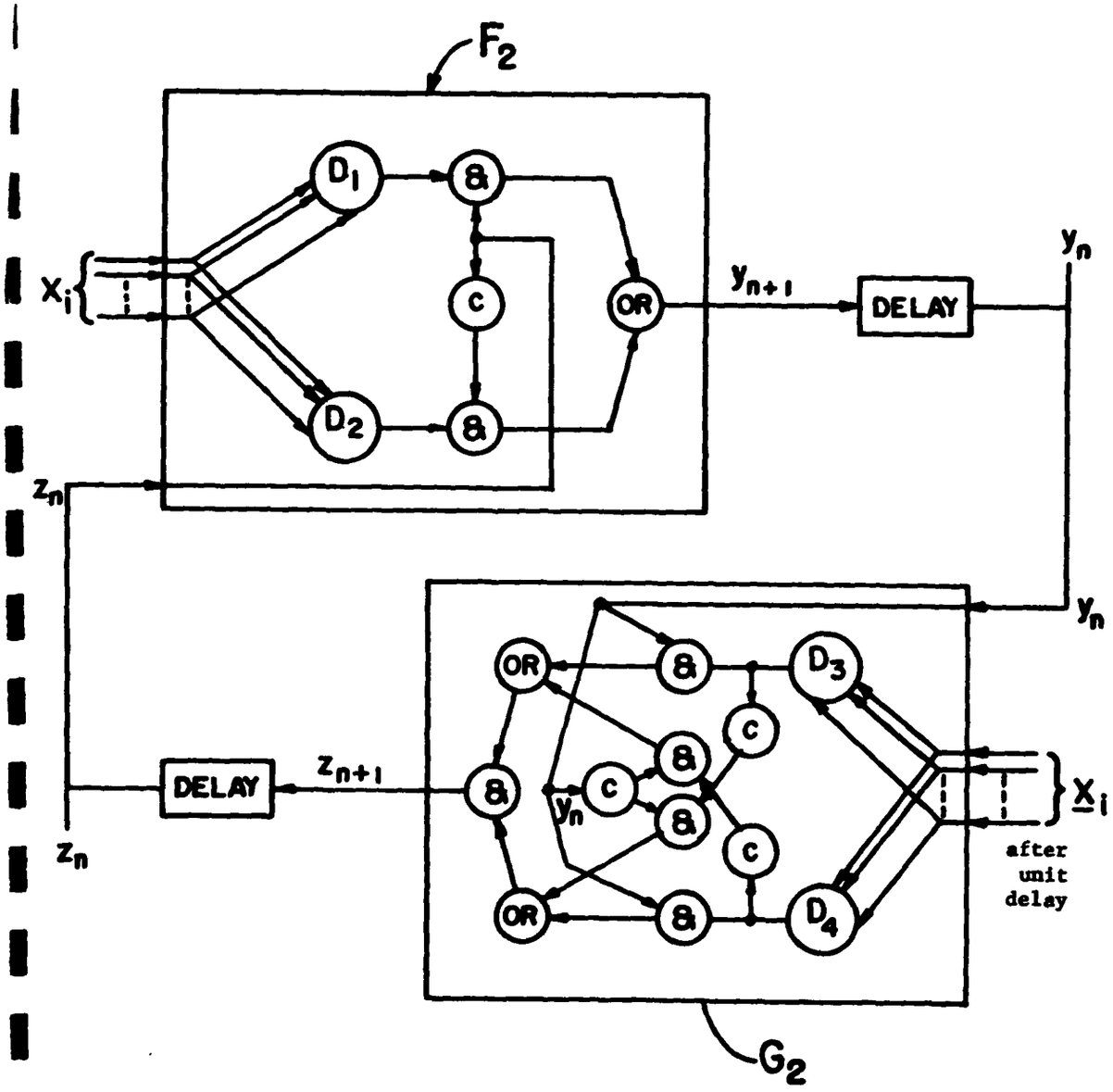


Fig. 13. The net, $\mathcal{F}_2$; a modified form of F_1 using greater redundancy in calculating z_n.

where ϵ_1 is defined in equation 41. One may notice that $\epsilon_B \ll \epsilon_A$. Thus, in this case, calculations in a computation are likely to be more often continued than be terminated erroneously.

In Figures 12 and 13 the feedback input, z, to the F-nets, actually selects an output from one of the two D nets, D_1 and D_2. This selection process involves the use of additional gates that reduce the first measure of reliability of the output*, y, from α at the output of the D-nets to nearly $(\alpha + 3\beta)$ at the output of the delay unit (reference to equations (39) to (41)). In any scheme involving the use of alternate nets the selection process of the outputs cannot be avoided. However, part of the selection may be done at the input end of the D-nets, as shown in Figure 14a. In this figure the input, z, is used to activate at a time only one of the two nets: D_1 when $z = 1$ and D_2 when $z = 0$. When a net is not activated its output may be assumed to be zero**. Thus the output of the 'OR' gate in Figure 14 a will be the same as the output of the net that had been activated. Or else, the scheme shown in Figure 14b may be used, where a switch has been employed to select the appropriate output. If the probability of both D_1 and D_2 being unactivated is γ and the probability of the switches (or the

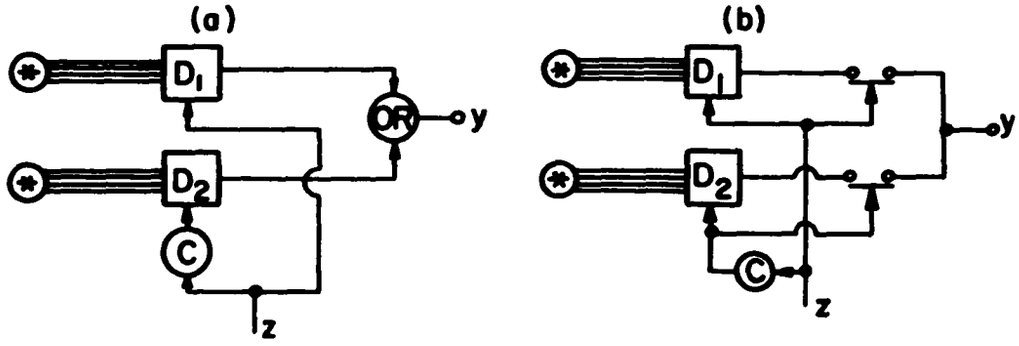


Fig. 14 Illustrating the use of z as an actuating signal.

---

* Note that the reliability of a net is considered to be the same as the reliability of its output.

** In a typical net, in practice, the activation of a net may be caused by providing the power supply necessary to operate a net. When no power is supplied to a net, its output has to be zero in the steady state.

OR gate) making a wrong selection is β, then the first measure of reliability of the output y will less than $(\alpha + \beta + \gamma)$. If more than one feedback line were available then more than one D net may be actuated in each calculation. In such a case, the output selection may be more profitably done by a majority gate instead of an OR gate or a switch. The use of a majority gate for output selection will result in a considerable improvement in reliability. Table III is a summary of the different probabilities of F and G units in Figures 12 and 13.

Table III: The Probabilities Associated with the Subnets of Figures 12 and 13.

Subnet	Probability
D_i	$p'(D_i) = \alpha$ for all i
Any control gate: &, OR, C, etc.	$p'(\&) = \beta$
F_1 of Fig. 12	$p'(F_1) \leq \epsilon_1 = \alpha + 3\beta$
G_1 of Fig. 12	$p'(G_1) \leq \epsilon_1 = \epsilon_A = \epsilon_\beta = \alpha + 3\beta$
F_2 of Fig. 13	$p'(F_2) \leq \epsilon_1 = \alpha + 3\beta$
G_2 of Fig. 13	$P\{z_{n+1} = 0 \mid y_n = f_D(\underline{X})\} \leq \epsilon_A = \epsilon_1(1-2\beta)(2-\epsilon_1) + \beta$
	$P\{z_{n+1} = 1 \mid y_n \neq f_D(\underline{X})\} \leq \epsilon_B = \epsilon_1^2(1-2\beta) + \beta$

4. Analysis of the Nets Described in Section 3.

The Markov chain diagram for the nets $\mathfrak{N}_1$ and $\mathfrak{N}_2$ of Figures 12 and 13 will be as shown in Figure 15. The transition probabilities, shown in Figure 15 are for the case: $f_D(\underline{X}) = 1$ and initial state, (0,1).

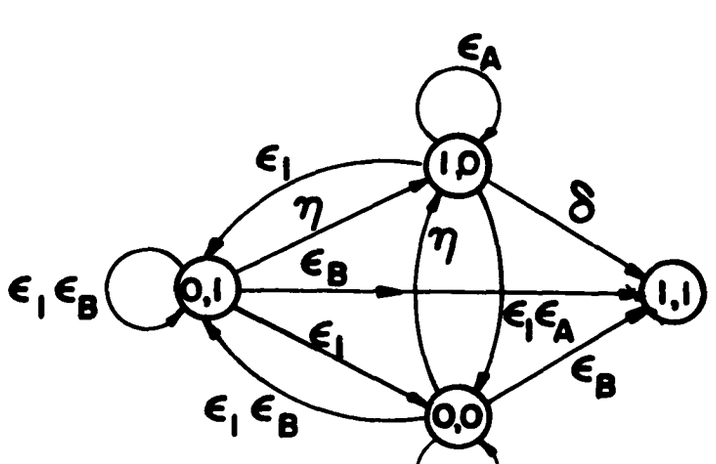


Fig. 15 Markov chain diagram for the nets $\mathfrak{F}_1$ and $\mathfrak{F}_2$ for the case: Initial state (0,1) and $f_1(\underline{X}_1) = 1$.

The probabilities ϵ_1, ϵ_A, ϵ_B in Figure 15 are given in Table III and η, δ are given by:

$$\eta = 1 - (\epsilon_1 + \epsilon_B + \epsilon_1 \cdot \epsilon_B) \tag{53}$$

and

$$\delta = 1 - (\epsilon_1 + \epsilon_A + \epsilon_1 \cdot \epsilon_A) \tag{54}$$

The probabilities η and δ indicate the correct transitions, as given in Table I, for the case, $f_D(\underline{X}) = 1$ (rows 1, 3, 5, and 7 of the table). The weight ϵ_1 indicates an erroneous transition -- not as required by Table I -- where the output, y, in the 'next state' is in error. The weights ϵ_A and ϵ_B indicate that the outputs z in the 'next states' are in error, the errors being of type A in one case, and type B* in the other. The weights $\epsilon_1 \cdot \epsilon_A$ and $\epsilon_1 \cdot \epsilon_B$ indicate that both y and z are in error in the 'next states' of the transitions. As the state, (1,1), in Figure 15 will always be a terminating state, no transitions are shown emanating from the state.

Using Table I and the error-probabilities shown in Table III the

* See Table III for definition of ϵ_A and ϵ_B.

Markov chain diagram for any initial state and output value may be similarly drawn. In Figure 15, for example, if the states (1,0), (0,0) are transposed and, so also, (1,1), (0,1) are transposed the resulting diagram will represent the case: $f_D(\underline{X}) = 0$ and initial state, (1,1). One may verify that the basic structure of the diagram for all possible initial and output states remains the same. Thus, the probabilities of proper, and erroneous computations calculated for Figure 15 will remain unchanged for other cases also.

Using the z-transform techniques and the analogy of signal flow graphs, one may obtain the following expression for the different statistical quantities associated with the Markov chain diagram of Figure 15. Let,

$$\sum_{\ell=2}^{\infty} P_p(\ell) \cdot z^{-\ell} = \hat{P}_p(z) \tag{55}$$

and

$$\sum_{\ell=2}^{\infty} P_e(\ell) \cdot z^{-\ell} = \hat{P}_e(z) . \tag{56}$$

Assuming that the length of each computation is ≥ 2, we get that,

$$\hat{P}_p(1) = P_p \tag{57}$$

and

$$\hat{P}_e(1) = P_e \tag{58}$$

For the Figure 15,

$$\hat{P}_p(z) = \frac{\eta\, z^{-1}\,(\delta + \epsilon_1)}{(1-\epsilon_1\, z^{-1})(1-\epsilon_A\, z^{1}) - \eta\, \epsilon_1\, \epsilon_A\, z^{-2}} \tag{59}$$

$$\hat{P}_e(z) = \frac{\epsilon_1\, z^{-1}\,(1-\epsilon_A\, z^{-1}(1-\eta))\,\epsilon_B(1+\epsilon_1)}{(1-\epsilon_1\, z^{-1})(1-\epsilon_A\, z^{-1}) - \epsilon_1\, \epsilon_A\, \eta\, z^{-2}} \tag{60}$$

$$E(\ell_p) = \frac{2 - \epsilon_1 - \epsilon_A}{1-\epsilon_1 - \epsilon_A + \epsilon_1\, \epsilon_A\,(1-\eta)} \tag{61}$$

$$V(\ell_p) = \frac{\epsilon_1 + \epsilon_2 - \epsilon_1 \epsilon_A (1-\eta)(4-\epsilon_1 - \epsilon_A)}{(1-\epsilon_1 - \epsilon_A + \epsilon_1 \epsilon_A (1-\eta))^2} \tag{62}$$

$$E(\ell_e) = 1 + \frac{1-\epsilon_A (1-\eta)(2-\epsilon_A)}{(1-\epsilon_B (1-\eta))(1-\epsilon_1 - \epsilon_A + \epsilon_1 \epsilon_A(1-\eta))} \tag{63}$$

$$V(\ell_e) = \frac{\epsilon_1 + \epsilon_A (5\eta-4) - 8\epsilon_A^2 (1-\eta)^2}{(1-\epsilon_B (1-\eta))^2 (1-\epsilon_1 - \epsilon_A + \epsilon_1 \epsilon_A(1-\eta))^2} \tag{64}$$

The values of $P_p(\ell)$ have been plotted in Figure 16 for various values of the parameters α and k where,

$$k = \alpha/\beta, \tag{65}$$

α and β are defined in Table III. In a sense, 'k' is a measure of the relative complexity of the D-nets in Figures 12 and 13 as compared to the individual control gates -- AND, OR or C. The larger the value of k is, one may say that the greater would be the complexity of the D-nets as compared to the individual AND, OR or C gates. One may notice that in the net, $\mathcal{F}_2$, the computations tend to be longer.

The values of P_p, P_e, $E(\ell_p)$ and $E(\ell_e)$ are tabulated in Table IV. One may notice that for k = 1, there is a loss in reliability for α = .05. When k = 10 the gain in reliability is ignificant. Also, in all cases the net, $\mathcal{F}_2$, gives greater improvement in reliability at the cost of increased average delay time for computation.

It may also be seen that in all cases $E(\ell_e) \geq E(\ell_p)$. This does not satisfy the condition given in inequality (30). It remains to be seen whether it is possible to satisfy (30) through the use of this scheme of self-correction, and if it is, what the requirements on the feedback net will be.

Table IV: Calculated Values of P_p, P_e, $E(\ell_p)$ and $E(\ell_e)$

α	k	P_p		P_e		$E(\ell_p)$		$E(\ell_e)$	
		1	2	1	2	1	2	1	2
0.05	1	.91095	.96743	.08905	.03257	2.5906	3.0050	2.4942	2.6846
	10	.9945	.99952	.0055	.00048	2.148	2.236	2.139	2.209
	100	.99967	.99996	.00033	.00004	2.114	2.117	2.108	2.1613
0.01	1	.99981	.999949	.00019	.000051	2.086	2.144	2.083	2.134
	10	.999983	.999986	$.17\times10^{-4}$	$.14\times10^{-5}$	2.026	2.041	2.026	2.040
	100	.999989	.999999	$.11\times10^{-4}$	$.10\times10^{-5}$	2.021	2.031	2.020	2.031
0.001	1	.999984	.999995	$.16\times10^{-4}$	$.41\times10^{-5}$	2.008	2.013	2.008	2.013
	10	.999998	.9999999	$.17\times10^{-5}$	$.13\times10^{-6}$	2.002	2.004	2.002	2.004
	100	.9999989	.99999999	$.11\times10^{-5}$	$.10\times10^{-7}$	2.002	2.003	2.002	2.003

The nets $\mathcal{F}_1$ and $\mathcal{F}_2$ of Figures 12 and 13, respectively, do not provide much protection against permanent failures. No redundant gates have been provided in the control network. As the control gates are simple AND, OR gates, one may introduce redundant gates through the 'triangulation' scheme of Levy[16] and Amarel.[15] Let us assume that such a scheme has been used and the probability of permanent failure of the control part of the F and G nets is, p"(c), in $\mathcal{F}_1$ and $\mathcal{F}_2$.

Let us now consider the failure of one of the D-nets. In the F-nets of both Figures 12 and 13 failure of either D_1 or D_2 will not cause a failure of the total F-net. The self-correcting feature -- namely, the substitution of an alternate net in place of an erroneous one -- will assure us of the correct output in each computation. Thus the F-nets of $\mathcal{F}_1$ and $\mathcal{F}_2$ will fail only if both D_1 and D_2 fail, or the control part of the net fails. Therefore,*

* p" is the second measure of unreliability.

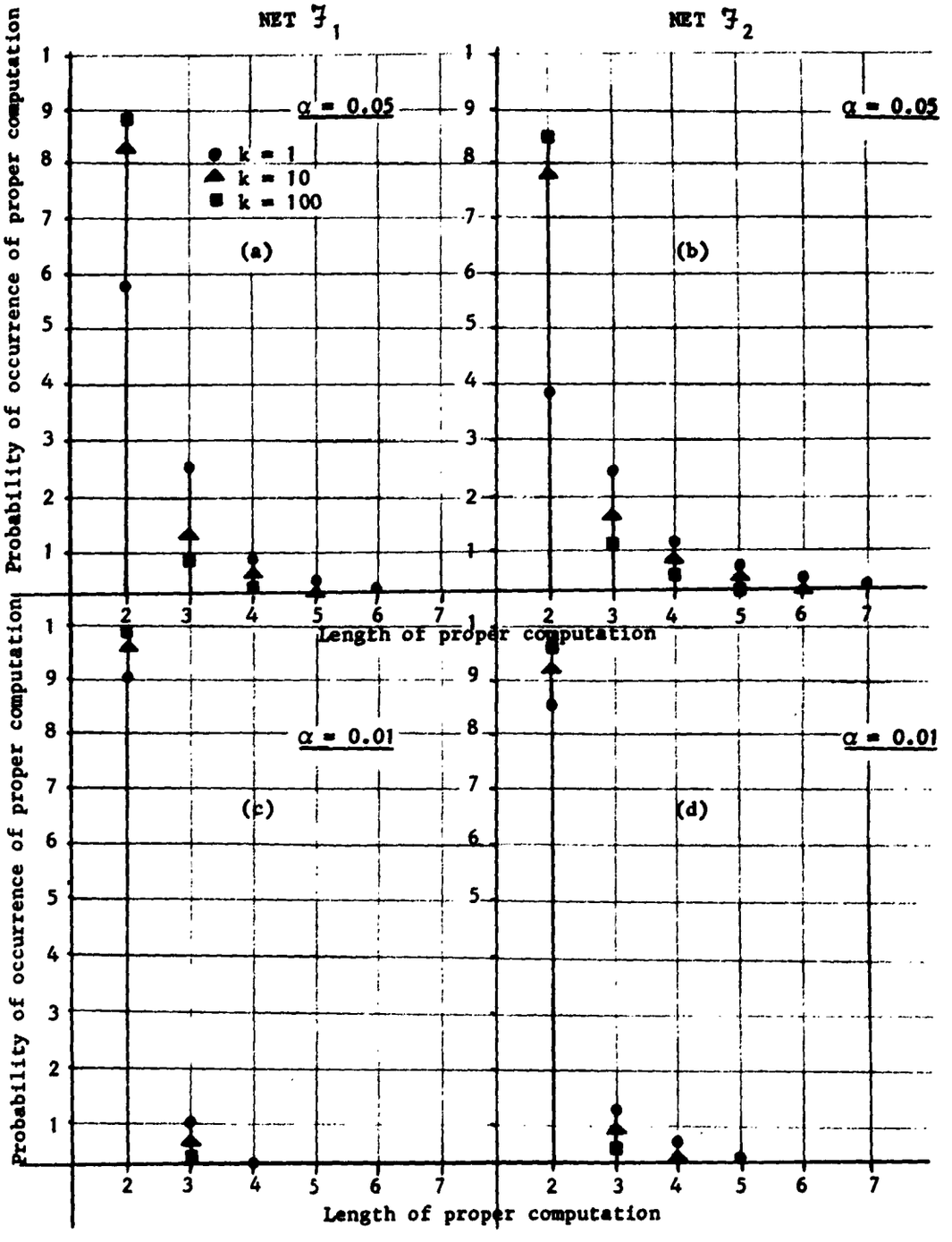


Fig. 16 Plots of probability of correct computation against lengths of the computations for various values of α and k, for the nets $\mathcal{F}_1$ and $\mathcal{F}_2$.

$$p''(F_i) = \{p''(D)\}^2 + p''(C),\ i = 1, 2 \tag{66}$$

for the nets $\mathfrak{F}_1$ and $\mathfrak{F}_2$,

In the case of the G-nets of $\mathfrak{F}_1$ and $\mathfrak{F}_2$, failure of either D_3 or D_4 will cause a failure of self-correcting property. Further, if z gets stuck at 0, no computation will ever terminate. So,

$$p''(G_i) = p''(D) + p''(C),\ i = 1, 2 \tag{67}$$

However, when D_3 fails the nets $\mathfrak{F}_1$ and $\mathfrak{F}_2$ will operate properly for the output y = 1, and also, when D_4 fails, proper operation will be available for y = 0. Unless z gets stuck at 0 the overall nets $\mathfrak{F}_1$ and $\mathfrak{F}_2$ may still be used, even though the self-correcting feature will be absent for some outputs. The probability of z getting stuck at 0 will be less than p"(D) + p"(C). Thus, some improvement will be obtained in the overall reliability of the nets $\mathfrak{F}_1$ and $\mathfrak{F}_2$. In general, one may choose a weighted sum of $p''(F_i)$ and $p''(G_i)$ for $p''(\mathfrak{F}_i)$, i = 1, 2:

$$p''(\mathfrak{F}_i) = c_{F_i} \cdot p''(F_i) + c_{G_i} \cdot p''(G_i) \tag{68}$$

for i = 1, 2, where $c_{F_i} = 1$ and $c_{G_i} < 1$, are constant coefficients.

The above discussion was intended only to illustrate the techniques of calculation and relative values of the various parameters that may be associated with a self-correcting net.

One may intuitively see that there exists a great variety of possibilities in which such nets may be constructed when the number of feedback lines is large. A greater understanding of the subject will have to be developed before a definite conclusion may be made about the practical applicability of the scheme of self-correction, introduced in this report.

REFERENCES

1. Esary, J. D., and F. Proschan, "The Reliability of Coherent Systems", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

2. Moore, E. F. and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays", Journal of the Franklin Institute, 262, 191-208 (September 1956), 281-297 (October 1956).

3. Von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, Annals of Mathematics Studies, No. 34, Princeton University Press, 43-49 (1956).

4. Blum, M., N. M. Onesto, and L. A. M. Verbeek, "Tolerable Errors of Neurons for Infallible Nets", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

5. Kautz, W. H., "Codes and Coding Circuitry for Automatic Error Correction within Digital Systems", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

6. Gore, W., "System Redundancy and Information Theory", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

7. Winograd, S., and J. D. Cowan, "Minimally Redundant Reliable Computing Systems", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

8. Peterson, W. W. "On Checking an Adder", IBM Journal of Research and Development, No. 2, 166-168 (April 1958).

9. Tryon, J. G., "Quadded Logic", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

   [9* See also Tryon, J. G. (to Bell Telephone Laboratories, Inc.) "Redundant Logic Circuitry", U. S. Patent 2,942,193 (Filed July 30, 1958), Saltzberg, B. R. (to Bell Telephone Laboratories, Inc.) "Redundant Logic Circuitry", U. S. Patent 3,016,517 (Filed May 15, 1959)].

10. Pierce, W. H., "Adaptive Vote Takers Improve the Use of Redundancy", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

11. Mann, W. C., "Restorative Processes for Redundant Computing Systems", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

12. Elias, P., "Computation in the Presence of Noise", IBM Journal of Research and Development, 2, No. 4, 346-53 (October 1958).

13. Peterson, W. W., and M. O. Rabin, "On Codes for Checking Logical Operations", IBM Journal of Research and Development, 3, No. 2, 63-68 (January 1958).

14. Winograd, S., "Coding for Logical Operations", IBM Journal of Research and Development, 6, No. 4, 430-436 (October 1962).

15. Amarel, S., and J. A. Brzozowski, "Theoretical Considerations on Reliability Properties of Recursive Triangular Networks", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

16. Levy, S. Y., "The Reliability of Recursive Triangular Switching Networks Built of Rectifier Gates", Redundancy Techniques for Computing Systems, Editors Wilcox and Mann, Washington, D. C., Spartan Books.

[15*, 16* See also Special Scientific Report No. 1, B. Reliability of Switching Networks, Air Force Contract AF19(604)-8423, April 1962].

17. Seshu, S., "Self-Repairing Machines", RADC-TR-61-91B, April, 1961; Syracuse University Research Institute, College of Engineering, Electrical Engineering Department, Report No. EE759-614F2.

18. Kaplan, K. R., "Analysis of Asynchronous Markov Chains", RCA Laboratories Internal Report, December 1962.

19. Huggins, W. H., "Signal Flow Graphs and Random Signals", Proc. IRE, Vol. 45, January 1957, pp. 74-86.

A. F. Cambridge Research Laboratories, Bedford, Mass., Rpt. No. AFCRL-63-185, THEORY OF ADJUSTABLE SWITCHING NETWORKS, Spec. Scient. Rpt. No. 2, April '63, 154 p. incl. illus., tables and refs.

Unclassified Report

Part 1 (Threshold Logic) contains a survey of threshold logic, a geometric result to estimate the number of threshold functions, generalizations and strengthening of known bounds on logical capabilities of threshold gate networks, computer-aided work on the realization of arbitrary functions by networks of 3-input majority gates, and a comparison of 2 methods for synthesis of very large threshold gates: The well-known Bayesian approach and a geometric alternative.

Part 2 (Reliability of Switching Networks) contains a survey of several important schemes for introducing redundancy into a combinational network for the improvement of reliability. Comparisons are made with the recursive triangle system, some extensions of the previous analyses of recursive triangles, and initial results on the incorporation of memory and feedback to allow the use of fewer basic gates in a time-shared fashion.

1. Threshold Logic
2. Artificial Intelligence
3. Bionics
4. Computer Logic
5. Switching Networks

I. AFSC Project 4641 Task 464101
II. Contract AF19(604)-8423
III. RCA Laboratories, Princeton, N.J.
IV. S. Amarel, S.Y. Levy, R. O. Winder & C. V. Srinivasan
V. Spec. Scient. Rpt. No. 2
VI. In DDC collection

A. F. Cambridge Research Laboratories, Bedford, Mass., Rpt. No. AFCRL-63-185, THEORY OF ADJUSTABLE SWITCHING NETWORKS, Spec. Scient. Rpt. No. 2, April '63, 154 p. incl. illus., tables and refs.

Unclassified Report

Part 1 (Threshold Logic) contains a survey of threshold logic, a geometric result to estimate the number of threshold functions, generalizations and strengthening of known bounds on logical capabilities of threshold gate networks, computer-aided work on the realization of arbitrary functions by networks of 3-input majority gates, and a comparison of 2 methods for synthesis of very large threshold gates: The well-known Bayesian approach and a geometric alternative.

Part 2 (Reliability of Switching Networks) contains a survey of several important schemes for introducing redundancy into a combinational network for the improvement of reliability. Comparisons are made with the recursive triangle system, some extensions of the previous analyses of recursive triangles, and initial results on the incorporation of memory and feedback to allow the use of fewer basic gates in a time-shared fashion.

1. Threshold Logic
2. Artificial Intelligence
3. Bionics
4. Computer Logic
5. Switching Networks

I. AFSC Project 4641 Task 464101
II. Contract AF19(604)-8423
III. RCA Laboratories, Princeton, N.J.
IV. S. Amarel, S.Y. Levy, R. O. Winder & C. V. Srinivasan
V. Spec. Scient. Rpt. No. 2
VI. In DDC collection

A. F. Cambridge Research Laboratories, Bedford, Mass., Rpt. No. AFCRL-63-185, THEORY OF ADJUSTABLE SWITCHING NETWORKS, Spec. Scient. Rpt. No. 2, April '63, 154 p. incl. illus., tables and refs.

Unclassified Report

Part 1 (Threshold Logic) contains a survey of threshold logic, a geometric result to estimate the number of threshold functions, generalizations and strengthening of known bounds on logical capabilities of threshold gate networks, computer-aided work on the realization of arbitrary functions by networks of 3-input majority gates, and a comparison of 2 methods for synthesis of very large threshold gates: The well-known Bayesian approach and a geometric alternative.

Part 2 (Reliability of Switching Networks) contains a survey of several important schemes for introducing redundancy into a combinational network for the improvement of reliability. Comparisons are made with the recursive triangle system, some extensions of the previous analyses of recursive triangles, and initial results on the incorporation of memory and feedback to allow the use of fewer basic gates in a time-shared fashion.

1. Threshold Logic
2. Artificial Intelligence
3. Bionics
4. Computer Logic
5. Switching Networks

I. AFSC Project 4641 Task 464101
II. Contract AF19(604)-8423
III. RCA Laboratories, Princeton, N.J.
IV. S. Amarel, S.Y. Levy, R. O. Winder & C. V. Srinivasan
V. Spec. Scient. Rpt. No. 2
VI. In DDC collection

A. F. Cambridge Research Laboratories, Bedford, Mass., Rpt. No. AFCRL-63-185, THEORY OF ADJUSTABLE SWITCHING NETWORKS, Spec. Scient. Rpt. No. 2, April '63, 154 p. incl. illus., tables and refs.

Unclassified Report

Part 1 (Threshold Logic) contains a survey of threshold logic, a geometric result to estimate the number of threshold functions, generalizations and strengthening of known bounds on logical capabilities of threshold gate networks, computer-aided work on the realization of arbitrary functions by networks of 3-input majority gates, and a comparison of 2 methods for synthesis of very large threshold gates: The well-known Bayesian approach and a geometric alternative.

Part 2 (Reliability of Switching Networks) contains a survey of several important schemes for introducing redundancy into a combinational network for the improvement of reliability. Comparisons are made with the recursive triangle system, some extensions of the previous analyses of recursive triangles, and initial results on the incorporation of memory and feedback to allow the use of fewer basic gates in a time-shared fashion.

1. Threshold Logic
2. Artificial Intelligence
3. Bionics
4. Computer Logic
5. Switching Networks

I. AFSC Project 4641 Task 464101
II. Contract AF19(604)-8423
III. RCA Laboratories, Princeton, N.J.
IV. S. Amarel, S.Y. Levy, R. O. Winder & C. V. Srinivasan
V. Spec. Scient. Rpt. No. 2
VI. In DDC collection